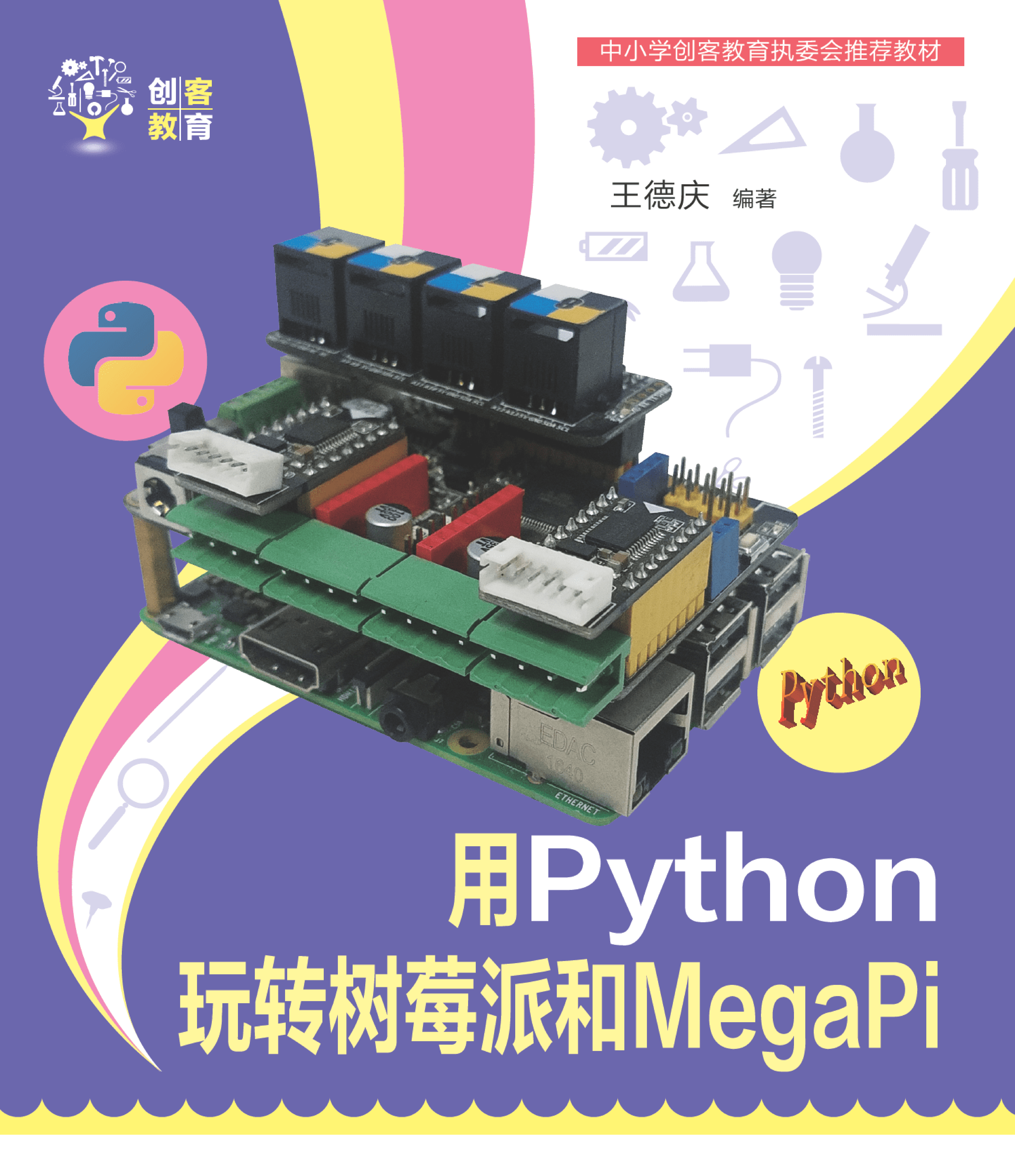
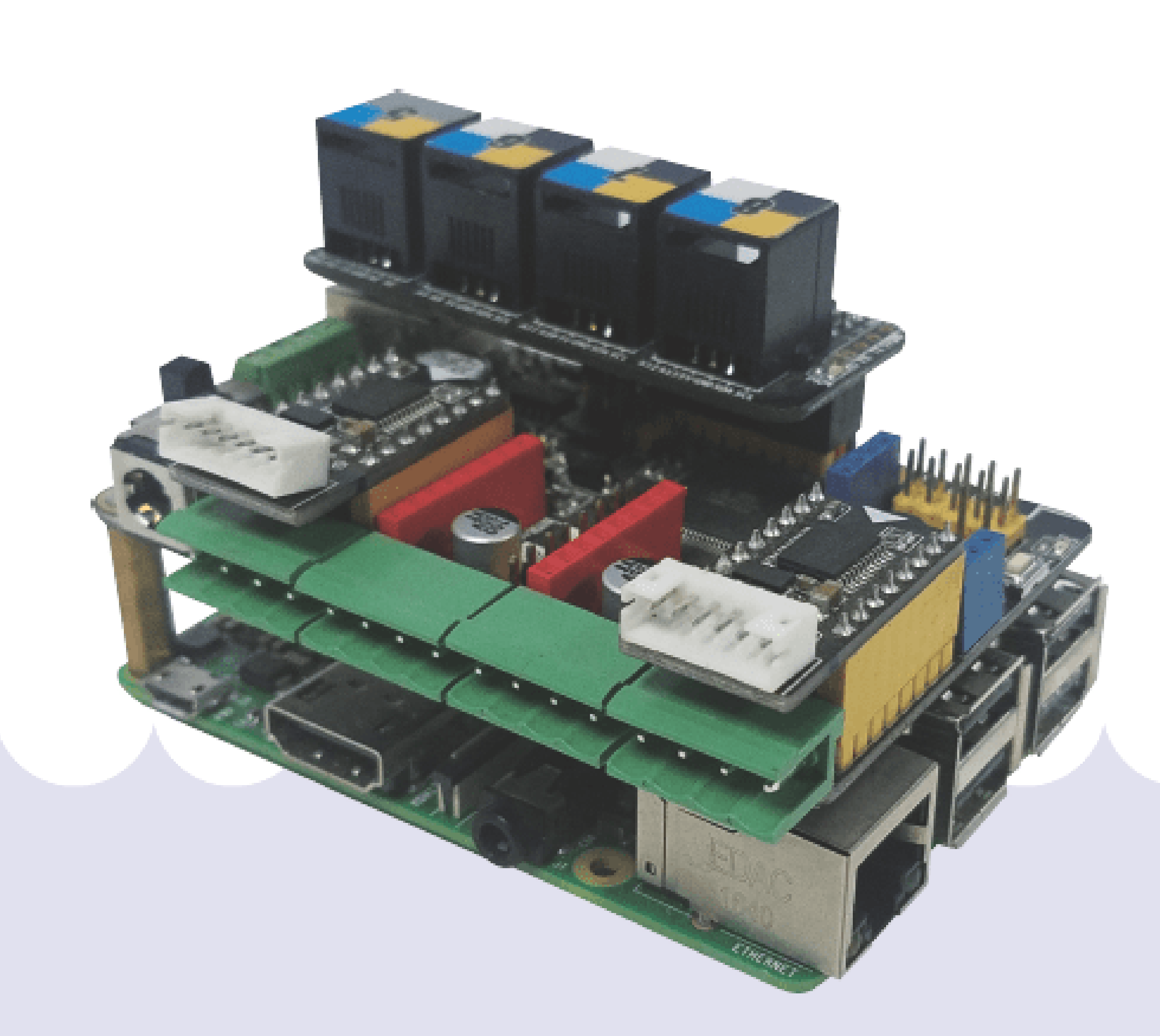


清华大学出版社





用Python 玩转树莓派和MegaPi

王德庆 编著

清华大学出版社
北 京

内 容 简 介

Python是免费、开源、功能强大的编程语言，树莓派和Arduino是主流的开源硬件。本书将三者结合，以Python编程语言为主线，详细介绍了将树莓派作为上位机，将基于Arduino的MegaPi作为下位机，来控制各种传感器、电动机等硬件设备，为创客提供了全新的解决方案。

本书适合已经有一定开源软硬件知识基础的中学生，可作为他们创客课程、机器人课程的教材使用，也可作为各种培训机构的教学参考用书。

图书在版编目（CIP）数据

用 Python 玩转树莓派和 MegaPi / 王德庆编著 . —北京：清华大学出版社，2019
（创客教育）
ISBN 978-7-302-52635-3

Ⅰ. ①用…　Ⅱ. ①王…　Ⅲ. ①软件工具—程序设计　Ⅳ. ① TP311.561

中国版本图书馆 CIP 数据核字（2019）第 046891 号

责任编辑： 王剑乔
封面设计： 傅瑞学
责任校对： 赵琳爽
责任印制： 丛怀宇

出版发行： 清华大学出版社
网　　址： http: //www.tup.com.cn, http: //www.wqbook.com
地　　址： 北京清华大学学研大厦A座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn
印 装 者： 北京嘉实印刷有限公司
经　　销： 全国新华书店
开　　本： 203mm × 260mm　　**印　　张：** 5.75　　**字　　数：** 139千字
版　　次： 2019年5月第1版　　**印　　次：** 2019年5月第1次印刷
定　　价： 39.00元

产品编号：082228-01

丛书编委会

序/

人人创客　创为人人

少年强则国强。风靡全球的创客运动一开始就与教育有着千丝万缕的联系。这种联系主要表现在两个方面：一是像 3D 打印、智能机器、创意美食等融合了“高大上”的最新科技和普通人可以操作的、方便快捷的东西，本身就有很强的吸引力，很多青少年是被其吸引过来而不是被叫过来的，这样自然意味着创客运动有很大的教育意义；二是创客运动对教育的更大挑战是，让这些青少年真正地面对真实社会。在自媒体的时代，信息传播的成本基本为零，任何一个人在任何一个年龄段都可以分享自己的创意，甚至这个创意还在雏形阶段，“未成形，先成名”。社交网络上的真诚点赞和可能带来的潜在商机，让投身创客学习模式的青少年在锻炼动手能力和创新思维的同时，找到了一个和社会直接对接的端口。

那么，一个好的创客应该具备什么样的品质呢？首先是“发现问题”，发现自己和身边人的任何一个微小需求，哪怕它很“偏门”，比如一个用来检测紫外线强度是否过强的帽子。但是根据“长尾理论”，有了互联网，世界各地的人们能够搜索到这种小众的发明，然后为其付费。其次是“质感品位”，做一个有设计思维的人，能够用设计师的方式去思考，当别人看到自己设计的东西时总有一种“工匠精神”之感——确实花了很多心思去设计，真诚地为自己点赞。也可以在开始时就有自己的品牌特色，比如设计一个商标或者统一外部特征。物像人一样，我们可以察觉到它们的不同个性，好的设计像一个富有个性的人一样有它的特色。通过欣赏好的设计，并且去制造它，可以提高自己对质感的把握能力和对品位的理解能力，使自己的创客作品能够超越“粗糙发明”的状态，成为一个精致的造物。再次是要能够驾驭价值规律，可以从很多现成的套件入手，但是最终一定要能够驾驭原始材料，如基础控制板、电子元器件、木头、塑料、铝等，因为只有这样才能驾驭成本。几乎没有小饭馆会采用从大酒店订餐然后再卖给自己顾客的做法，因为它们无法卖出大酒店的价格。同样，用现成套件搭建的作品也卖不出去，因为它的成本太高，现成套件只是一个很好的入门途径。通过一步步的学习，最终学会了驾驭原始材料，就能够实现物品的使用价值和成本之间的飞跃。就像我们用废旧物品制作机器人一样，它

仿佛在对你说："谢谢你给予了我新的生命，原来我一文不值，现在却成为大家眼里的明星。"而这种价值提升的过程也是创客特别引以为傲的地方。最后就是"资源和限制"，知道自己擅长什么、不擅长什么，才能很好地寻找合作伙伴，所有的创新都在有限资源和无限想象力之间"妥协"。通过了解物和人的资源及限制，就可以驾驭自己无限的想象力了。你肯定会想："哦，我明白了，创客就是对于任何一个自己或者别人微小的需求都能够用有质感和品位的方式来满足，从中得到价值上的提升，并且能够组建团队创造性地解决问题的一群人。"那么我会回答："嗯……我也不太清楚，因为创客领域的所有答案都要你亲自动手去解决，你先去做，然后告诉我，我说得对不对。""那么，我要怎么做呢？"

"创客教育"系列丛书提供了充分选择的空间，里面琳琅满目的创客项目，总有一款适合你。那么，亲爱的朋友，如果你现在能够对自己说，第一，我想学，而且如果一时找不到老师，我愿意自学；第二，我想去做一个快乐、自由的创造者，自己开心也能够帮助身边的人解决问题，那么你在思想上已经是一个很优秀的创客了。试想，一个"人人创客，创为人人"的社会应该是怎样的呢？我们认为一定是一个每个人都能够找到自己最愿意干的事，每个人都能够找到适合自己的项目"搭档"的世界。我们说得到底对不对呢？请大家动动手，亲自验证吧！

丛书编委会

2015年6月

前　言

MegaPi 是一款基于 ATmega2560 芯片的主控板。通过对驱动接口的良好封装，它可以快速简单地驱动编码电动机、直流电动机、步进电动机，还可以外接各种传感器。同时支持 Arduino IDE 和图形化编程。强大的运动控制能力和拓展性使 MegaPi 可以适用于 3D 打印机、CNC、创意设计和机器人等各种应用场景。

树莓派是一款基于 ARM 的微型电脑主板，外形只有信用卡大小，又称卡片式电脑，具备所有 PC 的基本功能，只须接通电视机和键盘，就能执行如电子表格、文字处理、玩游戏、播放高清视频等诸多功能。

Python 是纯粹的开源软件，已经成为最受欢迎的程序设计语言之一。Python 具有丰富和强大的库，同时由于 Python 语言的简洁性、易读性以及可扩展性，已经成为众多程序员的首选语言，并且已经成为多个省市中学的信息技术必修课程。

本书主要介绍 MegaPi 与树莓派的结合，树莓派用户使用 Python 就能实现对各种电动机及电子模块的控制。本书以知识内容划分章节，全书 4 章，前 3 章分别介绍 MegaPi 的各种端口、树莓派的安装与设置以及 Python 语言的编程基础知识；第 4 章重点介绍树莓派与 MegaPi 的结合，如何控制各种传感器和电动机。通过完成一系列有趣的项目制作，学习掌握树莓派、单片机、各种传感器、Python 编程、人工智能等各项知识。

本书力图拓展学生的视野，将学科知识与项目活动相结合，从而成为中小学生 STEM 教育课程的一个尝试与探索。

本书例程环境为 PC：Windows 7；Python：32 位 Python 3.6.1；Opencv：2.4.9.1；树莓派 3B+：操作系统 Raspbian。

MegaPi 及各种电动机和电子模块均由 Makeblock 公司提供。

本书的编写得到了郑剑春老师的大力帮助，得到了北京机器人教育领域多位老师的建议和意见，还得到清华大学出版社的大力支持和帮助，在此一并表示感谢。

作　者

2019 年 2 月

目　录

第 1 章 Python 基础

1.1 Python简介与安装

1.1.1 Python 简介

全世界有 600 多种编程语言，流行的编程语言有 20 余种。你可能已经听说过很多种流行的编程语言，比如 C 语言、C++、Java、Visual Basic、JavaScript 等。Python 也是一种计算机程序设计语言，是著名的“龟叔”Guido van Rossum 在 1989 年圣诞节期间，为了打发无聊的圣诞节而编写的一种编程语言。

目前，Python 已经成为最受欢迎的程序设计语言之一。2011 年 1 月，它被 TIOBE 编程语言排行榜评为“2010 年度语言”。自从 2004 年以后，Python 的使用率呈线性增长。

Python 语言是少有的一种可以称得上既简单又功能强大的编程语言。使用 Python 语言后你会惊喜地发现编程是多么简单，它注重的是如何解决问题而不是编程语言的语法和结构。

由于 Python 语言的简洁性、易读性以及可扩展性，在国外用 Python 做科学计算的研究机构日益增多，一些知名大学已经采用 Python 来教授程序设计课程。例如，卡耐基梅隆大学的编程基础、麻省理工学院的计算机科学及编程导论就使用 Python 语言讲授。许多大型网站也是用 Python 开发的，例如，YouTube、Instagram 以及豆瓣。很多大公司，包括 Google、Yahoo 等，甚至 NASA（美国航空航天局）都大量地使用 Python。

众多开源的科学计算软件包都提供了 Python 的调用接口，例如，著名的计算机视觉库 OpenCV、三维可视化库 VTK、医学图像处理库 ITK。而 Python 专用的科学计算扩展库就更多了，例如，这 3 个经典的科学计算扩展库 NumPy、SciPy 和 Matplotlib，它们分别为 Python 提供了快速数组处理、数值运算以及绘图功能。因此 Python 语言及其众多的扩展库

所构成的开发环境十分适合工程技术、科研人员处理实验数据、制作图表，甚至开发科学计算应用程序。

Python 适合开发的应用，首先是网络应用，包括网站、后台服务等；其次是许多日常需要的小工具，包括系统管理员需要的脚本任务等；另外，就是把其他语言开发的程序再包装起来，方便使用。本书主要介绍 Python 在智能机器人方面的应用。

Python 是开源的、免费的，网上有大量的资源，下面几个重要网址推荐给你使用，里面包含了大量的开发资源和源代码。

（1）https://www.python.org/

（2）https://www.python.org/doc/

（3）http://bugs.python.org/

（4）https://hackerone.com/python

（5）http://stackoverflow.com/questions/tagged/python

1.1.2　Python 的特点

Python 的设计哲学是“优雅”“明确”“简单”，即尽量写容易看明白的代码，尽量少写代码。

任何编程语言都有优缺点，Python 也不例外。Python 语言的优点如下。

（1）非常简单，非常适合阅读。阅读一个良好的 Python 程序感觉就像是在读英语，尽管这个“英语”的语法要求非常严格！ Python 的这种伪代码本质是它最大的优点。它使你专注于解决问题而不是去搞明白语言本身。

（2）易学。Python 虽然是用 C 语言写的，但它摈弃了 C 语言中非常复杂的指针，简化了 Python 的语法。

（3）Python 是 FLOSS（自由 / 开放源码软件）之一。简单地说，你可以自由地发布这个软件的拷贝，阅读它的源代码，对它做改动，把它的一部分用于新的自由软件中。

（4）可移植性。由于它的开源本质，Python 已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。如果你已经小心地避免使用 Python 中依赖于系统的特性，那么你的所有 Python 程序无须修改就可以在大部分的主流系统平台上运行。这些平台包括 Linux、Windows、FreeBSD、Macintosh、Solaris、OS/2、Amiga、AROS、AS/400、BeOS、OS/390、z/OS、Palm OS、QNX、VMS、Psion、Acom RISC OS、VxWorks、PlayStation、Sharp Zaurus、Windows CE，甚至还有 PocketPC、Symbian 以及 Google 基于 Linux 开发的 Android 平台。

（5）在计算机内部，Python 解释器把源代码转换成称为字节码的中间形式，然后再把它翻译成计算机使用的机器语言并运行。由于不再担心如何编译程序，如何确保连接转载正确的库等，这使得人们使用 Python 更加简单，只需要把 Python 程序复制到另外一台计算机上，它就可以工作了，这也使 Python 程序更加易于移植。

（6）Python 既支持面向过程的函数编程，也支持面向对象的抽象编程。在面向过程的语言中，程序是由过程或仅仅是可重用代码的函数构建起来的。在面向对象的语言中，程序是由数据和功能组合而成的对象构建起来的。与其他主要的语言如 C++ 和 Java 相比，Python 以一种非常强大又简单的方式实现面向对象编程。

（7）可扩展性和可嵌入性。如果需要自己的一段关键代码运行得更快或者希望某些算法不公开，你可以把这部分程序用 C 或 C++ 编写，然后在 Python 程序中使用它们。还可以把 Python 嵌入 C 或 C++ 程序，从而向程序用户提供脚本功能。

（8）丰富的库。Python 标准库很庞大，还有可定义的第三方库可以使用，它可以帮助你处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV 文件、密码系统、GUI（图形用户界面）、Tk 和其他与系统有关的操作。只要安装了 Python，所有这些功能都是可用的，这被称作 Python 的“功能齐全”理念。除了标准库以外，还有许多其他高质量的库，如 wxPython、Twisted 和 Python 图像库等。

（9）规范的代码。Python 采用强制缩进的方式使代码具有极佳的可读性。

Python 语言的缺点如下。

（1）运行速度慢。Python 是解释性语言。若有速度要求，可以用 C++ 改写关键部分。不过对于用户而言，机器上运行速度是可以忽略的，因为用户根本感觉不出这种速度的差异。

（2）单行语句和命令行输出问题：很多时候不能将程序连写成一行。

（3）Python 的开源性使 Python 语言不能加密。

（4）构架选择太多（没有像 C# 这样的官方 .net 构架，也没有像 Ruby 构架开发的相对集中）。不过这也从另一方面说明，Python 比较优秀，吸引的人才多，项目也多。

1.1.3　Windows 下 Python 开发环境搭建

Python 的开发环境很人性化，没有过多烦琐的配置，下载以后直接安装就可以使用，其操作步骤如下。

（1）下载一个 Python IDLE 程序安装包。



打开 Python 的官方网站 www.python.org，单击 Downloads 按钮，进入页面后，找到适合自己系统的版本安装包，注意 32 位和 64 位安装包的区别。

（2）找到刚下载的 Python 程序安装包，双击打开，运行安装程序。一般无须过多设置，直接单击“下一步”按钮，直至安装成功，单击“完成”按钮就可以了。

（3）从开始菜单找到 Python IDLE，即可启动 Python 集成化编程环境，并可以看到当前安装的 Python 的版本号，如图 1-1 所示。

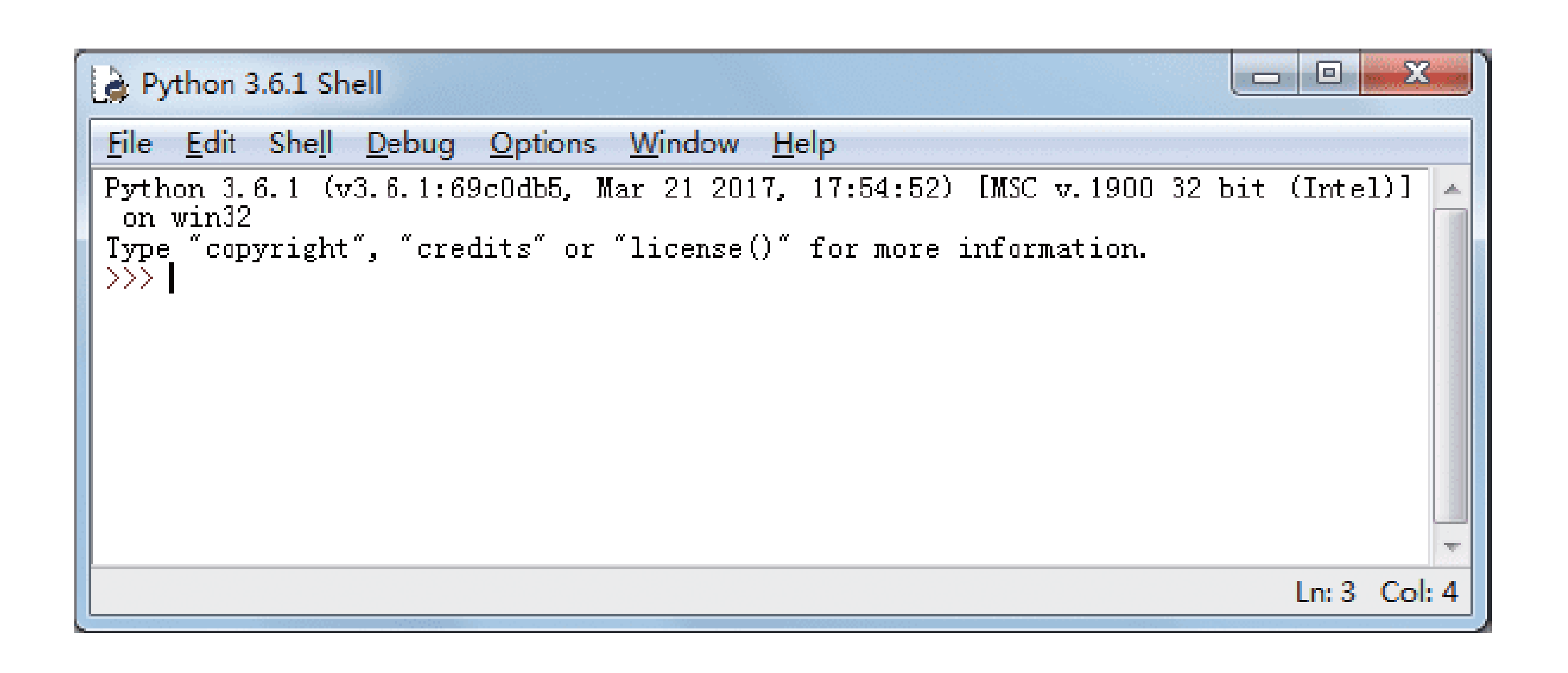


图1-1　当前安装的Python的版本号

除了默认安装的 IDLE，还有大量其他的 IDE 开发环境，例如 Wing IDE、PyCharm、PythonWin、Spyder 等。每个 IDE 都有不同的风格，也分别得到了不同开发人员的喜爱，但严格来说，这些开发环境都是对 Python 解释器 python.exe 的封装，核心是一样的，只是加了一个“外挂”而已，使用起来更加方便，减少了出错率，尤其是拼写错误。本书 Windows 环境下的 Python 开发环境选用官方标准开发环境 IDLE。

1.1.4　模块安装

Python 之所以得到青睐，与涉及各行业、各领域开发的扩展库有很大关系，它的扩展库不仅数量众多、功能强大，而且用起来很方便。虽然 Python 标准库已经提供了非常强大的功能，但很多时候若能够熟练运用扩展库，会事半功倍，大幅提高开发速度。

可以把 Python 模块看作是一个个用来存放积木的收纳箱，每个收纳箱里放着很多特定类型的积木（函数或类），用的时候，我们把收纳箱从仓库里拖出来，然后打开它，选择合适的积木搭建自己的房子、汽车、轮船、飞机等作品（程序）就可以了。

目前，pip 已经成为管理 Python 扩展库的主流方式。使用 pip 不仅可以实时查看本机已安装的 Python 扩展库列表，还支持 Python 扩展库的安装、升级和卸载等操作。使用 pip 工具管理 Python 扩展库只需要保证计算机联网的情况下，输入几个命令即可完成，极大地方便了用户。常用的 pip 命令使用方法如表 1-1 所示。

表 1-1　常用 pip 命令的使用方法

pip 命令示例	说　明
`pip freeze [> requirements.txt]`	以 requirements 的格式列出已安装模块
`pip list`	列出当前已安装的所有模块
`pip install SomePackage[==version]`	在线安装 SomePackage 模块的指定版本
`pip install SomePackage.whl`	通过 whl 文件离线安装扩展库
`pip install package1 package2...`	依次（在线）安装 package1、package2 等扩展模块
`pip install -r requirements.txt`	安装 requirements.txt 文件中指定的扩展库
`pip install -upgrade SomePackage`	升级 SomePackage 模块
`pip uninstall SomePackage[==version]`	卸载 SomePackage 模块的指定版本

使用 pip 命令安装 Python 扩展库需要在命令提示符环境中进行，并且需要切换至 pip 命令所在的目录。

Python 默认安装仅包含部分基本或核心模块，启动时也仅加载了基本模块，在需要的时候再加载其他模块，这样可以减小程序运行的压力，且具有很强的可扩展性，有助于提高系统的安全性。

安装完毕的模块，使用 import 命令导入，之后就可以使用模块中的函数或类了。

1.1.5　IDLE 简单应用

启动 IDLE 之后默认为交互模式，直接在 Python 提示符“>>>”后面输入相应的命令并按 Enter 键执行即可。如果顺利，马上就可以看到执行结果，否则会显示异常。例如：

```
>>> 3+5                      # 计算 3+5 的和
8
>>> 3*(2+6)                  # 计算表达式的值
24
>>>2**99                     # 计算 2 的 99 次方
633825300114114700748351602688
>>> print("hello world")   # 输出显示 "hello world"
hello world

>>> 2/0                      # 除数为 0，抛出异常
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    2/0
ZeroDivisionError: integer division or modulo by zero
>>>import math               # 导入 math 模块
```




```
>>>math.sin(0.5)            #求 0.5 弧度的正弦值
0.479425538604203
```

在 IDLE 界面中，选择菜单命令 File→New File，进入程序编辑模式，创建一个程序文件，输入代码并保存为 .py 或 .pyw 文件。

使用菜单命令 Run → Check Module 检查程序中是否存在语法错误，或者使用菜单命令 Run → Run Module 运行程序，程序运行结果将直接显示在 IDLE 交互界面上。

1.1.6　Python 文件名

.py：Python 源文件，由 Python 解释器负责解释执行。

.pyw：Python 源文件，常用于图形界面程序文件。

.pyc：Python 字节码文件，无法使用文本编辑器直接查看该类型文件内容，可用于隐藏 Python 源代码和提高运行速度。

Python 模块第一次被导入时，将被编译成字节码的形式，并在以后再次导入时优先使用 .pyc 文件，以提高模块的加载和运行速度。

1.2　Python基本语法

1.2.1　Python 的对象模型

对象是 Python 语言中最基本的概念，在 Python 中处理的一切都是对象。Python 中有许多内置对象可供编程者使用，内置对象可直接使用，如数字、字符串、列表、del 等；非内置对象需要导入模块才能使用，如要使用正弦函数 sin( )，需要导入 math 模块；要使用随机数产生函数 random( )，需要导入 random 模块等。常见的 Python 对象如表 1-2 所示。

表 1–2　常见的 Python 对象

对象类型	类型名称	示　　例	简 要 说 明
数字	int, float, complex	1234,3.14, 1.3e5, 3+4j	数字大小没有限制，内置，支持复数及其运算
字符串	str	'swfu', "I'm student", '''Python ''', r'abc', R'bcd'	使用单引号、双引号、三引号作为定界符，以字母 r 或 R 引导的表示原始字符串
字节串	bytes	b'hello world'	以字母 b 引导，可以使用单引号、双引号、三引号作为定界符

续表

对象类型	类型名称	示　　例	简 要 说 明
列表	list	`[1,2,3],['a', 'b', ['c', 2]]`	所有元素放在一对方括号中，元素之间使用逗号分隔，其中的元素可以是任意类型
字典	dict	`{1:'food',2:'taste', 3:'import'}`	所有元素放在一对大括号中，元素之间使用逗号分隔，元素形式为“键 : 值”
元组	tuple	`(2, -5, 6), (3,)`	所有元素放在一对圆括号中，元素之间使用逗号分隔，如果元组中只有一个元素，后面的逗号不能省略

1.2.2　Python 的注释

一个好的、可读性强的程序一般包含 30% 以上的注释。常用的注释方式主要有以下两种。

（1）以 # 开始，表示本行 # 之后的内容为注释。

例如：

```
# 这是一个注释
print("Hello, World!")
```

（2）包含在一对三引号 '''...''' 或 """...""" 之间且不属于任何语句的内容将被解释器认为是注释。

例如：

```
'''
这是多行注释，用三个单引号
这是多行注释，用三个单引号
这是多行注释，用三个单引号
'''
print("Hello, World!")
```

1.2.3　Python 变量

在 Python 中，不需要事先声明变量名及其类型，直接赋值即可创建各种类型的对象变量。这一点适用于 Python 任意类型的对象。

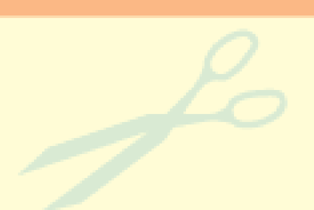
例如：

```
>>> x = 3        # 创建了整型变量 x，并赋值为 3
```

x → 3

图1–2　变量创建示意图

变量创建示意如图 1-2 所示。



```
>>> x = 'Hello world.'     # 创建了字符串变量 x，并赋值为 'Hello world.'
```

Python 属于强类型编程语言，Python 解释器会根据赋值或运算自动推断变量类型。Python 还是一种动态类型语言，变量的类型也可以随时变化。

```
>>> x = 3
>>> print(type(x))          # 查看变量类型
<class'int'>                # 整数类型

>>> x = 'Hello world.'
>>> print(type(x))          # 查看变量类型
<class'str'>                # 字符串类型

>>> x = [1,2,3]
>>> print(type(x))          # 查看变量类型
<class'list'>               # 列表类型

>>> isinstance(3, int)      # 测试对象是否是某个类型的实例
True

>>> isinstance('Hello world', str)
True
```

在 Python 中，允许多个变量指向同一个值，例如：

```
>>> x = 3
>>> id(x)
1786684560                  # 变量 x 的内存地址

>>> y = x
>>> id(y)                   # 变量 y 的内存地址
1786684560
```

然而，当为其中一个变量修改值以后，其内存地址将会变化，但这并不影响另一个变量，例如，接着上面的代码继续执行下面的代码：

```
>>> x += 6
>>> id(x)
1786684752

>>> y
3
>>> id(y)
1786684560
```

Python 采用的是基于值的内存管理方式，如果为不同变量赋值为相同值，这个值在内存中只有一份，多个变量指向同一块内存地址，如图 1-3 所示。

例如：

```
>>> x = 3
>>> id(x)
10417624

>>> y = 3
>>> id(y)
10417624
>>> x = [1, 1, 1, 1]
>>> id(x[0]) == id(x[1])
True
```

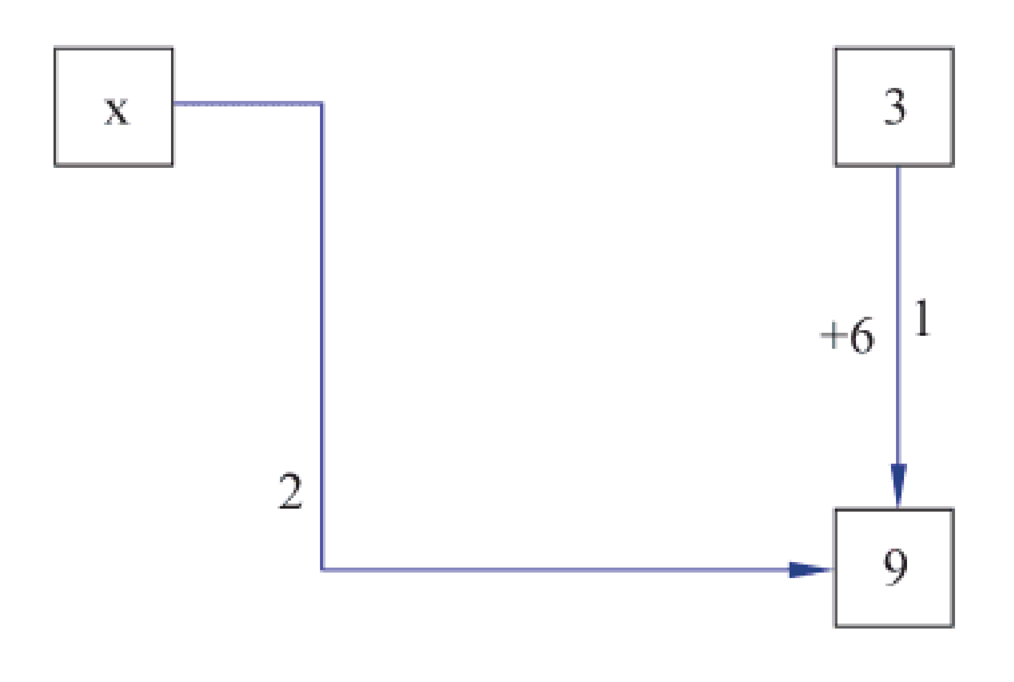


图1–3　内在地址示意图

Python 具有自动内存管理功能，对于没有任何变量指向的值，Python 自动将其删除。因为 Python 会跟踪所有的值，并自动删除不再有变量指向的值。所以 Python 程序员一般情况下不需要太多考虑内存管理的问题。

在定义变量名时，需要注意以下问题。

（1）变量名必须以字母或下划线开头，但以下划线开头的变量在 Python 中有特殊含义。

（2）变量名中不能有空格以及标点符号（括号、引号、逗号、斜线、反斜线、冒号、句号、问号等）。

（3）不能使用关键字作变量名，可以导入 keyword 模块后使用 print(keyword.kwlist) 查看所有 Python 的关键字。

（4）不建议使用系统内置的模块名、类型名或函数名以及已导入的模块名及其成员名作变量名，这将会改变其类型和含义，可以通过 dir(_ _builtins_ _) 查看所有内置模块、类型和函数。

（5）变量名对英文字母的大小写敏感，例如，student 和 Student 是不同的变量。

1.2.4　Python 数字

数字是不可变对象，可以表示任意大小的数字。例如：

```
>>> a=99999999999999999999999999999
>>> a*a
9999999999999999999999999999800000000000000000000000000001
>>> a**3   #a 的三次方
```






```
    999999999999999999999999999999997000000000000000000000000000000002
999999999999999999999999999999999
```

Python 的 IDEL 交互界面可以当作简便计算器使用。例如：

```
>>> ((3**2) + (4**2)) ** 0.5
5.0
```

Python 中的整数类型可以分为如下几种。

（1）十进制整数，如 0、–1、9、123。

（2）十六进制整数，需要 16 个数字或字符 0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f 表示整数，必须以 0x 开头，如 0x10、0xfa、0xabcdef。

（3）八进制整数，只需要 8 个数字 0、1、2、3、4、5、6、7 表示整数，必须以 0o 开头，如 0o35、0o11。

（4）二进制整数，只需要 2 个数字 0、1 表示整数，必须以 0b 开头，如 0b101、0b100。

（5）浮点数又称小数，如 15.0、0.37、–11.2、1.2e2、314.15e–2。

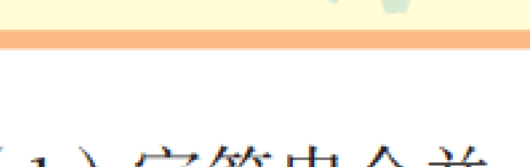
（6）Python 内置支持复数类型，例如：


```
>>> a = 3+4j
>>> b = 5+6j
>>> c = a+b
>>> c
(8+10j)
>>> c.real                    # 查看复数实部
8.0
>>> c.imag                    # 查看复数虚部
10.0
>>> a.conjugate()             # 返回共轭复数
(3-4j)
>>> a*b                       # 复数乘法
(-9+38j)
>>> a/b                       # 复数除法
(0.6393442622950819+0.03278688524590165j)
```

1.2.5 Python 字符串

用单引号、双引号或三引号括起来的符号系列称为字符串。

单引号、双引号、三单引号、三双引号可以互相嵌套，用来表示复杂字符串，例如：'abc'、'123'、' 中国 '、"Python"、'''Tom said, "Let's go"''' 都是符合语法的字符串。



（1）字符串合并。

```
>>> a = 'abc' + '123' #生成新对象
>>> a
'abc123'
```

（2）字符串格式化。

```
>>> a = 3.6674
>>> '%7.3f' % a          #保留 3 位小数，共 7 位，不足 7 位前面补空格
'3.667'
>>> "%d:%c" % (65,65)
'65:A'
```

常用的字符串格式符如下。

%s：获取传入对象的 __str__ 方法的返回值，并将其格式化到指定位置。

%c：将数字转换成 unicode 对应的值，将字符添加到指定位置。

%d：将整数、浮点数转换成十进制表示，并将其格式化到指定位置。

%a.bf：将整数、浮点数转换成浮点数表示，并将其格式化到指定位置（默认保留小数点后 6 位），a 表示浮点数的打印长度，b 表示浮点数小数点后面的精度。



1.2.6 Python 运算符和表达式

在 Python 中你会写大量的表达式。表达式由运算符和操作数组成，例如，2+3 就是一个表达式。常用的运算符见表 1-3。

表 1–3 常用的运算符

运 算 符	功 能 说 明
+	算术加法，列表、元组、字符串合并与连接，正号
–	算术减法，集合差集，相反数
*	算术乘法，序列重复
/	算术除法
//	求整商，但如果操作数中有实数，结果为实数形式的整数
%	求余数，字符串格式化
**	幂运算
<、<=、>、>=、==、!=	（值）大小比较，集合的包含关系比较
or	逻辑或
and	逻辑与
not	逻辑非

续表

<table>
<tr><th>运　算　符</th><th>功 能 说 明</th></tr>
<tr><td>in</td><td>成员测试</td></tr>
<tr><td>is</td><td>对象同一性测试，即测试是否为同一个对象或内存地址是否相同</td></tr>
<tr><td>|、^、&、<<、>>、~</td><td>位或、位异或、位与、左移位、右移位、位求反</td></tr>
<tr><td>&、|、^</td><td>集合交集、并集、对称差集</td></tr>
<tr><td>@</td><td>矩阵相乘运算符</td></tr>
</table>

Python 运算符优先级遵循的规则：算术运算符优先级最高，其次是位运算符 / 集合运算符、关系运算符、逻辑运算符，算术运算符之间遵循“先乘除，后加减”的基本运算原则。虽然 Python 运算符有一套严格的优先级规则，但是强烈建议在编写复杂的表达式时，尽量使用圆括号明确说明其中的逻辑，以提高代码的可读性。

1.2.7　基本输入 / 输出

用 Python 进行程序设计，输入是通过 input( ) 函数实现的，input( ) 的一般格式为

```
x = input('提示：')
```

该函数返回字符串类型的输入对象，可输入数字、字符串和其他任意类型对象。

例如：

```
>>> x=input("请输入 x：")
请输入 x：4              # 执行结果
>>> type（x）
<class 'str'>          #x 为字符串类型
```

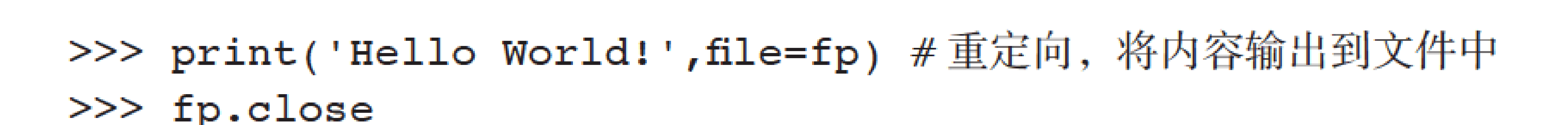
内置函数 print（ ）用于输出特定信息，语法格式为

```
print(value, ..., sep= '  ', end= '\n', file=sys.stdout, flush=False)
```

其中，sep 参数之前为需要输出的内容（可以有多个）;sep 参数用于指定数据之间的分隔符，默认为空格；end 参数用于指定行末的结束符，默认为换行；file 参数用于指定输出位置，默认为标准控制台，也可以重定向输出到文件。

例如：

```
>>> print(1,2,3,4,sep='\t') # 修改默认分隔符
1    2     3     4
>>> fp=open('D:\\test.txt','a+')
```



```
>>> print('Hello World!',file=fp) #重定向，将内容输出到文件中
>>> fp.close
```

1.2.8 Python 代码规范

（1）缩进。

① 类定义、函数定义、选择结构、循环结构，行尾的冒号表示缩进的开始。

② Python 程序依靠代码块的缩进来体现代码之间的逻辑关系，缩进结束就表示一个代码块结束了。

③ 同一个级别的代码块的缩进量必须相同。

④ 一般而言，以 4 个空格为基本缩进单位。

（2）每个 import 只导入一个模块。

（3）如果一行语句太长，可以在行尾加上“\”来换行分成多行，但是建议使用括号来包含多行内容。

（4）必要的空格与空行。

① 运算符两侧、函数参数之间、逗号两侧建议使用空格分开。

② 不同功能的代码块之间、不同的函数定义之间建议增加一个空行以增加可读性。

1.3 Python程序结构控制

结构控制（Structure Control）是一种程序运行的逻辑。Python 语言一共有三种控制结构：顺序结构、选择结构和循环结构。

（1）从执行方式上看，从第一条语句到最后一条语句完全按顺序执行，是简单的顺序结构。

（2）如果在程序执行过程中，根据用户的输入或中间结果去执行若干不同的任务，则为选择结构。

（3）如果在程序的某处，需要根据某项条件重复地执行某项任务若干次，直到满足或不满足某条件为止，这就构成循环结构。

大多数情况下，程序都不会是简单的顺序结构，而是顺序、选择、循环三种结构的复杂组合。



1.3.1　Python 条件表达式

在选择结构和循环结构中，条件表达式的值只要不是 False、0（或 0.0、0j 等）、空值 None、空列表、空元组、空集合、空字典、空字符串、空 range 对象或其他空迭代对象，Python 解释器均认为与 True 等价。从这个意义上来讲，几乎所有的 Python 合法表达式都可以作为条件表达式，包括含有函数调用的表达式。

关于表达式和运算符的内容上一节已经做了介绍，这里不再赘述，只重点介绍比较特殊的几个运算符。首先是关系运算符，与多数语言不同的是，在 Python 中的关系运算符是可以连续使用的，例如：

```
>>> print（1<2<3）
True
>>> print（1<2>3）
False
>>> print（1<3>2）
True
```


在 Python 中，条件表达式中不允许使用赋值运算符“=”，避免了其他语言中误将关系运算符“==”写作“=”带来的麻烦。

比较特殊的运算符还有逻辑运算符 and 和 or 以及关系运算符，具有短路求值或惰性求值的特点，只计算必须计算的表达式的值。以 and 为例，对于表达式“表达式 1 and 表达式 2”而言，如果“表达式 1”的值为 False 或其他等价值时，不论“表达式 2”的值是什么，整个表达式的值都是 False，此时“表达式 2”的值无论是什么都不影响整个表达式的值，因此将不会被计算，从而减少不必要的计算和判断。

在设计条件表达式时，如果能够大概预测不同条件失败的概率，并将多个条件根据 and 和 or 运算的短路求值特性进行组织，可以大幅度提高程序运行效率。

1.3.2　单分支选择结构

语句格式：

```
if 表达式：
    语句块
```

程序流程如图 1-4 所示。

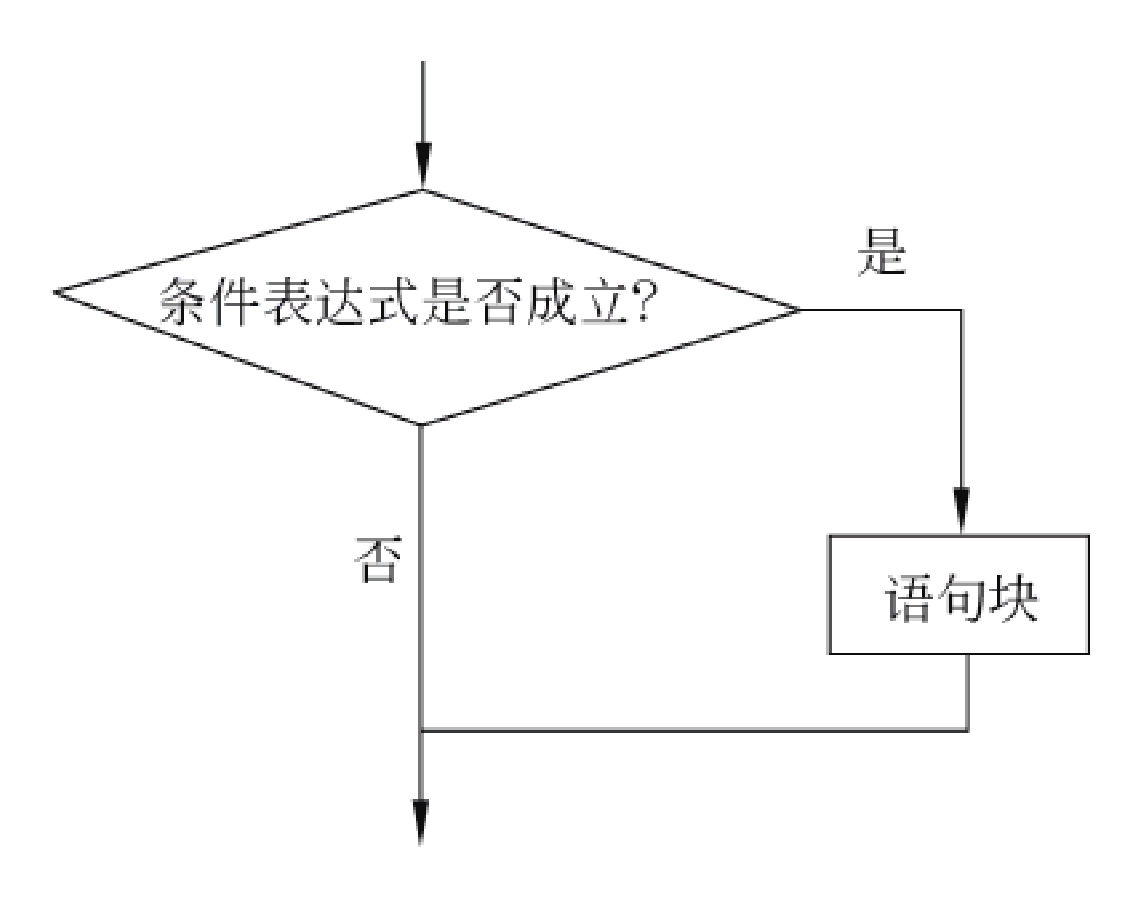


图1-4　单分支选择结构示意

例如，按照条件交换 2 个变量的值：

```
a=3
b=5
if a > b:
 a, b = b, a              # 交换两个变量的值
print(a, b)
```

1.3.3　双分支选择结构

语句格式：

```
if 表达式：
      语句块 1
else：
      语句块 2
```

程序流程如图 1-5 所示。

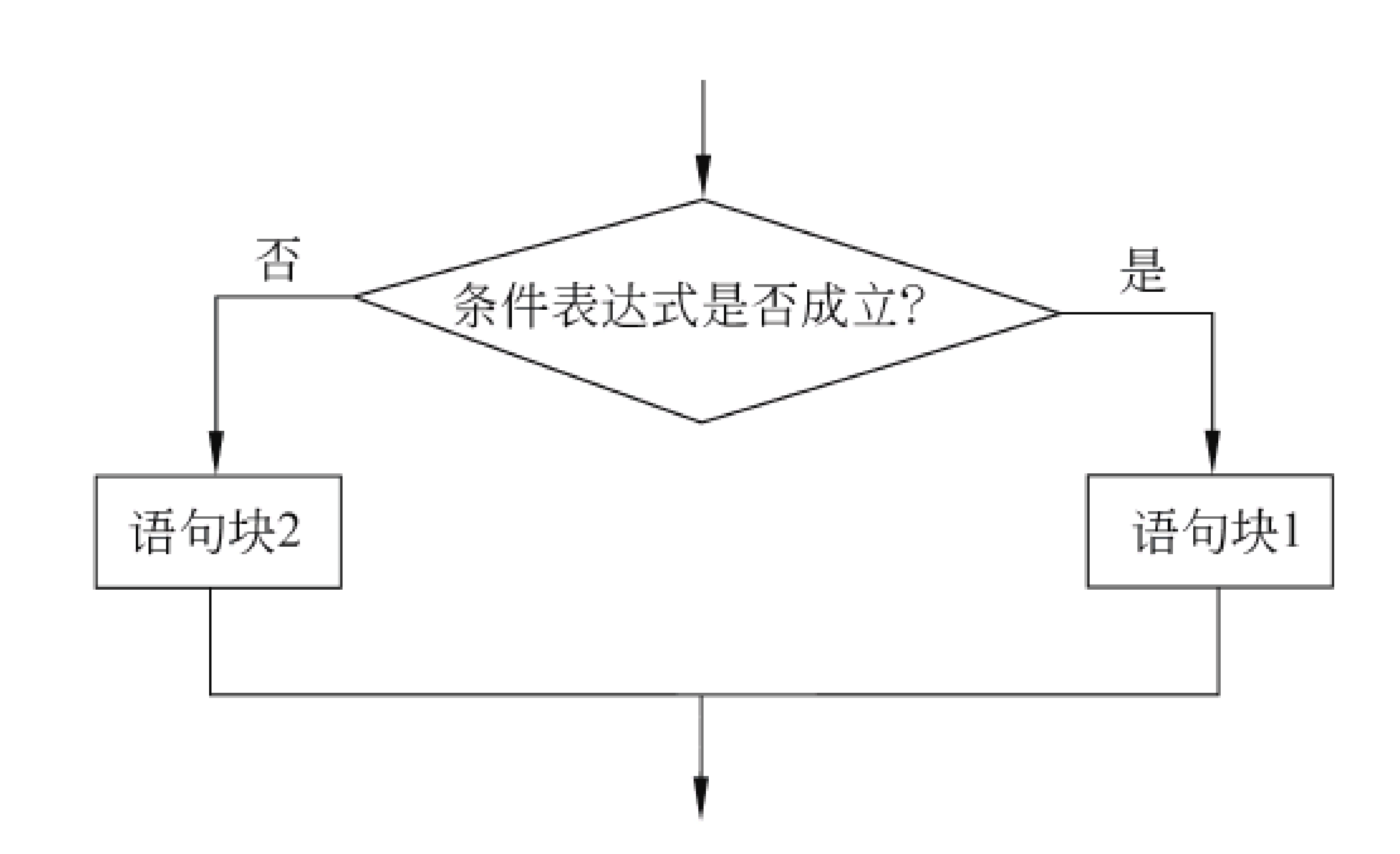


图1-5　双分支选择结构示意

例如，下面的双分支选择程序：

```
>>> chTest = ['1','2','3','4','5']
>>> if chTest:
      print(chTest)
else:
      print('Empty')
['1','2','3','4','5']
```

1.3.4　for 循环

for 循环一般用于循环次数可以提前确定的情况，尤其是用于枚举序列或迭代对象中的元素，一般优先考虑使用 for 循环。

语句格式：

```
for 取值 in 序列或迭代对象：
     循环体
[else：
     else 子句代码块 ]
```

例 1：计算 1+2+3+…+100 的值。

```
s=0
for i in range(1,101):     # 序列范围 [1,100]，步长为 1
      s = s + i
print('1+2+3+…+100 = ', s)
```

例 2：计算 1+3+5+…+99 的值。

```
s=0
for i in range(1,100,2):  #序列范围[1,99]，步长为 2
      s = s + i
print('1+3+5+…+99 = ', s)
```

例 3：求 1~100 之间能被 7 整除，但不能同时被 5 整除的所有整数。

```
for i in range(1,101):
      if (i % 7 == 0) and (i % 5 != 0):
           print(i)
```

1.3.5 while 循环

while 循环一般用于循环次数难以提前确定的情况，也可以用于循环次数确定的情况。

语句格式：

```
while 条件表达式：
     循环体
[else：
     else 子句代码块 ]
```

例 1：计算 1+2+3+…+100 的值。

```
s=0
i=0
while i<=100：
     s=s+i
     i=i+1
else:
     print(s)
```

为了优化程序以获得更高的效率和运行速度，在编写循环语句时，应尽量减少循环内部不必要的计算，将与循环变量无关的代码尽可能地提取到循环之外。对于使用多重循环嵌套的情况，应尽量减少内层循环中不必要的计算，尽可能地向外提。

1.3.6 break 和 continue 语句

break 语句在 while 循环和 for 循环中都可以使用，一般放在 if 选择结构中，一旦 break 语句被执行，将使整个循环提前结束。

continue 语句的作用是终止当前循环，并忽略 continue 之后的语句，然后回到循环的

顶端，提前进入下一次循环。

除非 break 语句让代码更简单或更清晰，否则不要轻易使用。

例 1：计算小于 100 的最大素数，注意 break 语句和 else 子句的用法。

```
for n in range(100, 1, -1):
    for i in range(2, n):
        if n%i == 0:
            break
    else:
        print(n)
        break
```

运行结果：97。

例 2：输出 10 以内的奇数。

```
for i in range(10):
    if i%2 == 0:
        continue
    print(i, end = '  ')
```

运行结果：1 3 5 7 9。

例 3：输出由 1、2、3、4 这 4 个数字组成的每位数都不相同的所有三位数。

```
digits = (1,2,3,4)
for i in digits:
    ii = i*100
    for j in digits:
        if j == i:
            continue
        jj = j * 10
        for k in digits:
            if k == i or k == j:
                continue
            print(ii + jj + k)
```

1.4　Python函数与类

函数是组织好的、可重复使用的、用来实现单一或相关联功能的代码段。

函数能提高应用的模块性和代码的重复利用率。Python 提供了许多内置函数，比如 print( )、input( ) 等。内置函数不需要导入任何模块即可直接使用。

可以使用 dir(_ _builtins_ _) 命令列出所有内置函数；可以使用 help(函数名) 查看某个函数的用法。

下面主要介绍 time 模块和 turtle 模块中的比较常用的函数。

1.4.1　日期和时间模块

Python 提供了一个 time 模块，可以用于格式化日期和时间，使用时需要 import 导入。

时间间隔是以秒为单位的浮点小数。每个时间戳都以自 1970 年 1 月 1 日午夜（历元）经过了多长时间来表示。

Python 的 time 模块下有很多函数可以转换常见的日期格式，如函数 time.time( ) 用于获取当前时间戳。

例 1：获取当前时间戳。

```
>>>import time
>>>print(time.time())
1508312234.7298932
```


时间戳最适合做日期运算，其返回的是从 1970 年 1 月 1 日起至当前的秒数。但是 1970 年之前的日期就无法以此表示了，未来太遥远的日期也不行，UNIX 和 Windows 只支持到 2038 年。

例 2：获取格式化的时间。

```
>>>import time
>>>print(time.asctime(time.localtime(time.time())))
Fir Aug 17 16:03:24 2018
```

其中，Fir 为星期五，Aug 为八月的缩写，后面依次为 17 日、时间和年。

还可以使用 time 模块的 strftime 方法格式化日期。

Python 中时间日期格式化符号如下。

（1）%y：两位数的年份表示（00~99）。

（2）%Y：四位数的年份表示（0000~9999）。

（3）%m：月份（01~12）。

（4）%d：月内中的一天（01~31）。

（5）%H：24 小时制小时数（00~23）。

（6）%I：12 小时制小时数（01~12）。

（7）%M：分钟数（00~59）。



（8）%S：秒（00~59）。

（9）%a：本地简化的星期名称。

（10）%A：完整的星期名称。

（11）%b：本地简化的月份名称。

（12）%B：完整的月份名称。

（13）%c：本地相应的日期表示和时间表示。

（14）%j：年内的一天（001~366）。

（15）%p：本地 A.M. 或 P.M. 的等价符。

（16）%U：一年中的星期数（00~53），星期天为星期的开始。

（17）%w：星期（0~6），星期天为星期的开始。

（18）%W：一年中的星期数（00~53），星期一为星期的开始。

（19）%x：本地相应的日期表示。

（20）%X：本地相应的时间表示。

（21）%Z：时区的名称。

例 3：获取格式化的日期或时间。

```
>>>import time
>>>print(time.strftime('%Y',time.localtime()))          # 获取完整年份
2017
>>>print(time.strftime('%y',time.localtime()))          # 获取简写年份
17
>>>print(time.strftime('%m',time.localtime()))          # 获取月
10
>>>print(time.strftime('%d',time.localtime()))          # 获取日
18
>>>print(time.strftime('%Y-%m-%d',time.localtime()))    # 获取年 - 月 - 日
2017-10-18
>>>print(time.strftime('%H',time.localtime()))          # 获取时,24 小时制
16
>>>print(time.strftime('%I',time.localtime()))          # 获取时,12 小时制
4
>>>print(time.strftime('%M',time.localtime()))          # 获取分
33
>>>print(time.strftime('%S',time.localtime()))          # 获取秒
31
>>>print(time.strftime('%H:%M:%S',time.localtime()))    # 获取时：分：秒
16:34:28
>>>print(time.strftime('%a',time.localtime()))          # 本地简化星期
```

```
Wed
>>>print(time.strftime('%A',time.localtime()))  # 本地完整星期
Wednesday
>>>print(time.strftime('%b',time.localtime()))  # 本地简化月份
Oct
>>>print(time.strftime('%B',time.localtime()))  # 本地完整月份
October
>>>print(time.strftime('%c',time.localtime()))  # 本地日期和时间表示
Wed Oct 18 16:37:46 2017
>>>print(time.strftime('%j',time.localtime()))  # 一年中的第几天
291
>>>print(time.strftime('%U',time.localtime()))  # 一年中的第几个星期，
                                                  星期天为星期的开始
42
>>>print(time.strftime('%w',time.localtime()))  # 星期几，星期天为星
                                                  期的开始
3
>>>print(time.strftime('%W',time.localtime()))  # 一年中的第几个星期，
                                                  星期一为星期的开始
42
>>>print(time.strftime('%x',time.localtime()))  # 本地日期表示
10/18/17
>>>print(time.strftime('%X',time.localtime()))  # 本地时间表示
17:16:17
>>>print(time.strftime('%Z',time.localtime()))  # 当前时区
CST
>>>print(time.strftime('%Y-%m-%d %H:%M:%S %w-%Z',time.localtime()))
2017-10-18 17:25:39 3-CST                        # 完整日期,时间,星期,
                                                  时区
```

1.4.2 turtle 模块

turtle 库是 Python 语言中一个很流行的绘制图像的函数库，想象一个小乌龟，在一个横轴为 x、纵轴为 y 的坐标系原点 (0,0) 位置开始，它根据一组函数指令的控制，在这个平面坐标系中移动，从而在它爬行的路径上绘制图形。

turtle 绘图的基础知识如下所述。

1. 画布 (canvas)

画布就是 turtle 为我们展开用于绘图的区域，我们可以设置它的大小和初始位置。

设置画布大小：

```
turtle.screensize(canvwidth=None, canvheight=None, bg=None)
```

其中，参数 canvwidth、canvheight、bg 分别为画布的宽（单位像素）、高和背景颜色。

例如：

```
turtle.screensize(800,600, "green")
turtle.screensize()          #返回默认大小(400, 300)
turtle.setup(width=0.5, height=0.75, startx=None, starty=None)
```

其中，参数 width、height 输入宽和高为整数时，表示像素；为小数时，表示占据计算机屏幕的比例；(startx, starty) 这一坐标表示矩形窗口左上角顶点的位置，如果为空，则窗口位于屏幕中心。

例如：

```
turtle.setup(width=0.6,height=0.6)
turtle.setup(width=800,height=800, startx=100, starty=100)
```

2. 画笔

在画布上，默认有一个坐标原点为画布中心的坐标轴，坐标原点上有一只面朝 *x* 轴正方向的小乌龟。这里我们描述小乌龟时使用了两个词语：坐标原点（位置）、面朝 *x* 轴正方向（方向）。在 turtle 绘图中，就是使用位置方向描述小乌龟（画笔）的状态。

画笔设置命令如下。

turtle.pensize()：设置画笔的宽度。

turtle.pencolor()：没有参数传入，返回当前画笔颜色；传入参数设置画笔颜色，可以是字符串如 "green"、"red", 也可以是 RGB 3 元组。

turtle.speed(speed)：设置画笔移动速度，画笔绘制的速度范围为 [0,10] 整数，数字越大，移动速度越快。

操纵小乌龟绘图有许多命令，这些命令可以划分为 3 种：运动命令、画笔控制命令和全局控制命令。

```
turtle.forward(distance)     #笔方向移动 distance 像素长度
turtle.backward(distance)    #向当前画笔相反方向移动 distance 像素长度
turtle.right(degree)         #顺时针移动 degree°
turtle.left(degree)          #逆时针移动 degree°
turtle.pendown()             #移动时绘制图形，默认时也为绘制
turtle.goto(x,y)             #将画笔移动到坐标为 (x,y) 的位置
turtle.penup()               #提起笔移动，不绘制图形，用于另起一个地方绘制
turtle.circle()              #画圆,半径为正(负),表示圆心在画笔的左边(右边)
                              画圆
setx()                       #将当前 x 轴移动到指定位置
sety()                       #将当前 y 轴移动到指定位置
```

```
setheading(angle)             # 设置当前朝向为 angle 角度
home()                        # 设置当前画笔位置为原点，朝向东
dot(r)                        # 绘制一个指定直径和颜色的圆点
turtle.fillcolor(colorstring) # 绘制图形的填充颜色
turtle.color(color1, color2)  # 同时设置 pencolor=color1, fillcolor=color2
turtle.filling()              # 返回当前是否在填充状态
turtle.begin_fill()           # 准备开始填充图形
turtle.end_fill()             # 填充完成
turtle.hideturtle()           # 隐藏画笔的 turtle 形状
turtle.showturtle()           # 显示画笔的 turtle 形状
turtle.clear()                # 清空 turtle 窗口,但是 turtle 的位置和状态不会改变
turtle.reset()                # 清空窗口，重置 turtle 状态为起始状态
turtle.undo()                 # 撤销上一个 turtle 动作
turtle.isvisible()            # 返回当前 turtle 是否可见
stamp()                       # 复制当前图形
turtle.write(s [,font=("font_name",font_size,"font_type")])
```

其中，s 为文本内容；font 为字体的参数，分别为字体名称、大小和类型；font 为可选项，font 参数也是可选项。

（1）turtle 绘图实例 1：太阳花。程序如下。

```
import turtle
import time
turtle.color("red", "yellow")  # 同时设置 pencolor = color1, fillcolor = color2
turtle.begin_fill()
for _ in range(50):
      turtle.forward(200)
      turtle.left(170)
turtle.end_fill()
turtle.mainloop()
```

（2）turtle 绘图实例 2：五角星。程序如下。

```
import turtle
import time
turtle.pensize(5)
turtle.pencolor("yellow")
turtle.fillcolor("red")
turtle.begin_fill()
for _ in range(5):
      turtle.forward(200)
      turtle.right(144)
turtle.end_fill()
time.sleep(2)
turtle.penup()
```

```
turtle.goto(-150,-120)
turtle.color("violet")
turtle.write("Done", font=('Arial',40,'normal'))
turtle.mainloop()
```

1.4.3　Python 定义函数

定义一个函数简单的规则如下。

（1）函数代码块以 def 关键词开头，后接函数标识符名称和圆括号 ()。

（2）任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。

（3）函数的第一行语句可以选择性地使用文档字符串（用于存放函数说明）。

（4）函数内容以冒号起始，并且缩进。

（5）return [表达式] 结束函数，选择性地返回一个值给调用方。如果函数没有 return 语句，或者有 return 语句但是没有执行，或者只有 return 而没有返回值，Python 将认为该函数以 return None 结束。

函数定义语法：

```
def 函数名([参数列表]):
函数体
```

默认情况下，参数值和参数名称是按函数声明中定义的顺序匹配起来的。

例 1：自定义函数示例。

```
def printme(str):                    # 定义函数
    print str

printme("我要调用用户自定义函数!")     # 调用函数
printme("再次调用同一函数")
```

以上程序的输出结果：

```
我要调用用户自定义函数!
再次调用同一函数
```

例 2：生成斐波那契数列的函数定义和调用。

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

```
fib(1000)                                # 调用函数
```

例 3：编写函数计算圆的面积。

```
from math import pi as PI
def CircleArea(r):
    if isinstance(r, (int,float)):    # 确保接收的参数为数值
            return PI*r*r
    else:
            print('You must give me an integer or float as radius.')
print(CircleArea(3))                     # 调用函数
```

1.4.4 变量的作用域

一个程序的所有的变量并不是在哪个位置都可以访问的。访问权限决定于这个变量是在哪里赋值的。变量的作用域决定了在哪一部分程序你可以访问哪个特定的变量名称。两种最基本的变量是全局变量和局部变量，定义在函数内部的变量拥有一个局部作用域，定义在函数外的拥有全局作用域。

局部变量只能在其被声明的函数内部访问，而全局变量可以在整个程序范围内访问。调用函数时，所有在函数内声明的变量名称都将被加入作用域中。

例 1：变量作用域示例。

```
total = 0                                # 这是一个全局变量
def sum(arg1, arg2):                     # 定义函数，功能是返回 2 个参数的和
    total = arg1 + arg2                  #total 在这里是局部变量
    print(" 函数内是局部变量 : ", total)
    return total
sum(10, 20)                              # 调用 sum 函数
print " 函数外是全局变量 : ", total
```

输出结果如下：

```
函数内是局部变量：30。
函数外是全局变量：0。
```

1.4.5 面向对象程序设计

Python 完全采用了面向对象程序设计的思想，是真正面向对象的高级动态编程语言，完全支持面向对象的基本功能，如封装、继承、多态以及对基类方法的覆盖或重写。

Python 中对象的概念很广泛，Python 中的一切内容都可以称为对象，除了数字、字符串、列表、元组、字典、集合、range 对象、zip 对象等，函数也是对象，类也是对象。

创建类时，用变量形式表示的对象属性称为数据成员，用函数形式表示的对象行为称为成员方法，成员属性和成员方法统称为类的成员。

Python 使用 class 关键字定义类，class 关键字之后是一个空格，然后是类的名字，再然后是一个冒号，最后换行并定义类的内部实现。

类名的首字母一般要大写，当然也可以按照自己的习惯定义类名，但一般推荐参考惯例来命名，并在整个系统的设计和实现中保持风格一致。

类定义基本语法：

```
class ClassName：
    '类的帮助信息 '             #类文档字符串
    class_suite                 #类体
```

类的帮助信息可以通过 ClassName.__doc__ 查看。

class_suite 由类成员、方法、数据属性组成。

例如：定义一个类。

```
class Car:                      #定义 Car 类
      def infor(self):          #定义 infor 方法
          print("This is a car")
```

定义了类之后，可以用来实例化对象，并通过“对象名 . 成员”的方式来访问其中的数据成员或成员方法。

```
>>> car = Car()
>>> car.infor()
  This is a car
```

面向对象编程中的概念很多，涉及的方面很广，很多都不好理解，这里只是介绍最基本的概念和方法，方便读者对后续知识内容的学习和理解。



第 2 章 MegaPi 基础

2.1 MegaPi简介

2.1.1 MegaPi 介绍

MegaPi 是一款为创客设计的主控板，同时也可以用于教育领域。MegaPi 基于 Arduino Mega 2560，该开发板主要分为 6 个功能区域，可以分别用来接入各式各样的外接模块，比如电动机、传感器和无线通信等。由于设计上的优势，MegaPi 可以非常方便地驱动各种电动机，它可以同时驱动 10 个舵机和 8 个 DC 发动机。因此 MegaPi 很适合做各种各样的机器人项目，例如智能机器人和 3D 打印机。

MegaPi 主控板实物如图 2-1 所示。

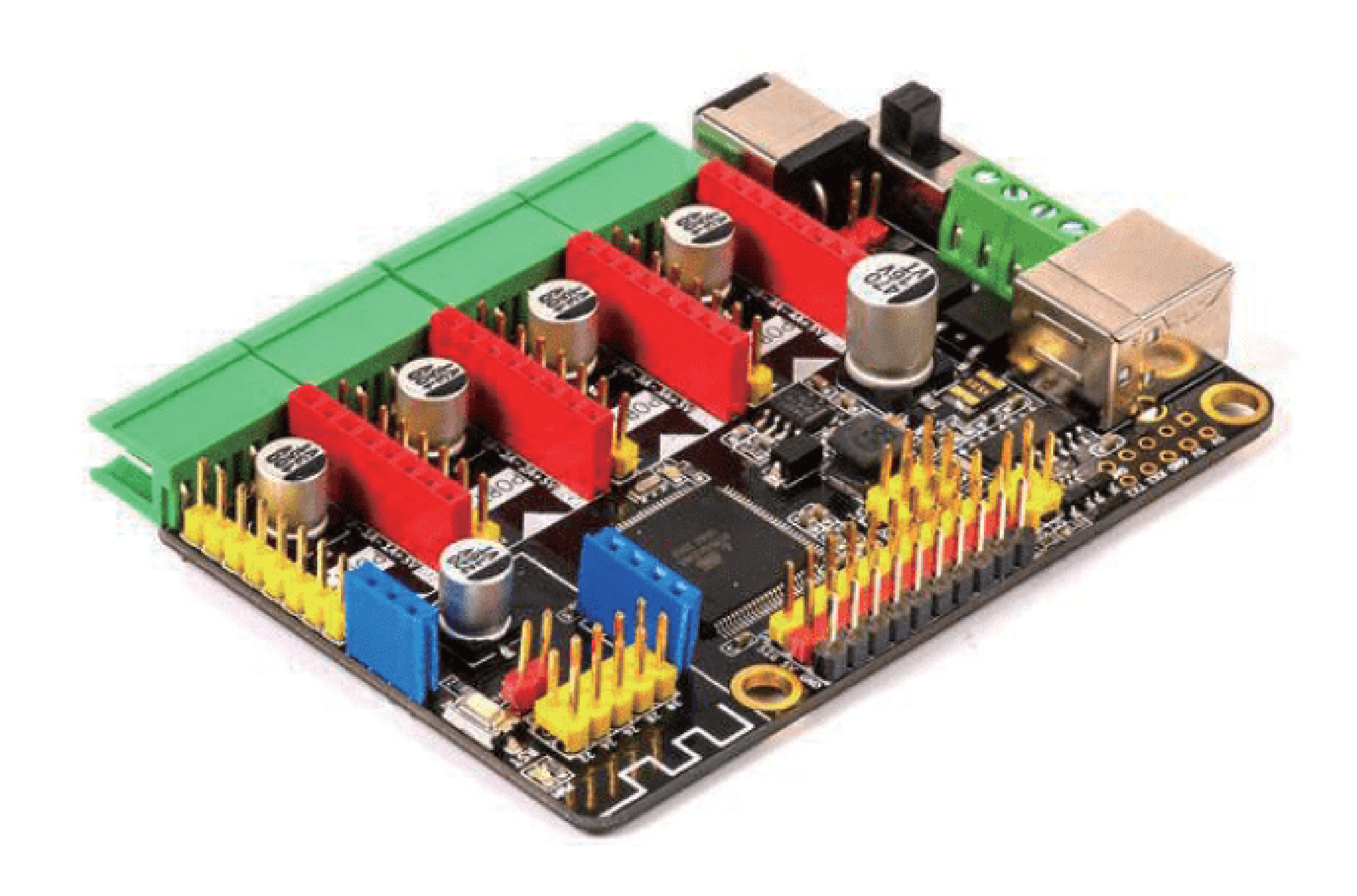


图2-1　MegaPi主控板



MegaPi 的主要硬件资源如下。

（1）4 个电动机驱动接口，用来接编码电动机驱动、步进电动机驱动，进而控制各种电动机。

（2）1 个无线通信接口，用来接蓝牙模块或者 2.4G 无线通信模块。

（3）10 个舵机接口，可以同时驱动 10 个舵机。

（4）2 个大功率 MOS 驱动，可以驱动最大 10A 的设备，I/O 端口最大为 5V、3A。

（5）一个可以连接树莓派的接口。

（6）一个 B 类型 USB 接口，可以用来下载程序。

MegaPi 连接电动机的实物图如图 2-2 所示。

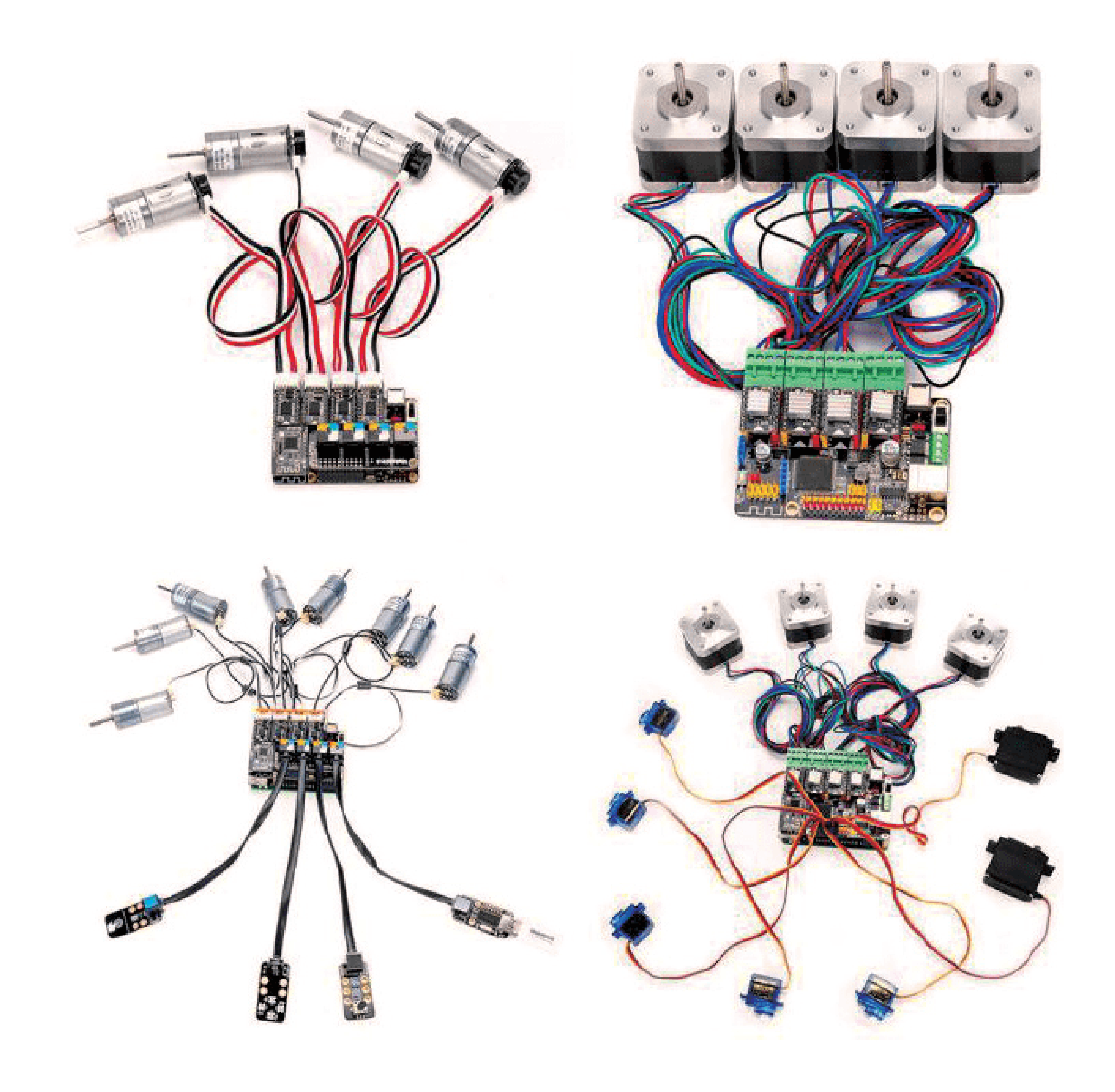


图2-2　MegaPi连接电动机

2.1.2　Megapi 功能区划分

MegaPi 提供了丰富的接口，如图 2-3 所示。

MegaPi 基于 Arduino Mega 2560，将 Arduino Mega 2560 的针脚做了封装和重新定义。图 2-4 列出了 MegaPi 和 Arduino Mega 2560 的针脚对应关系。



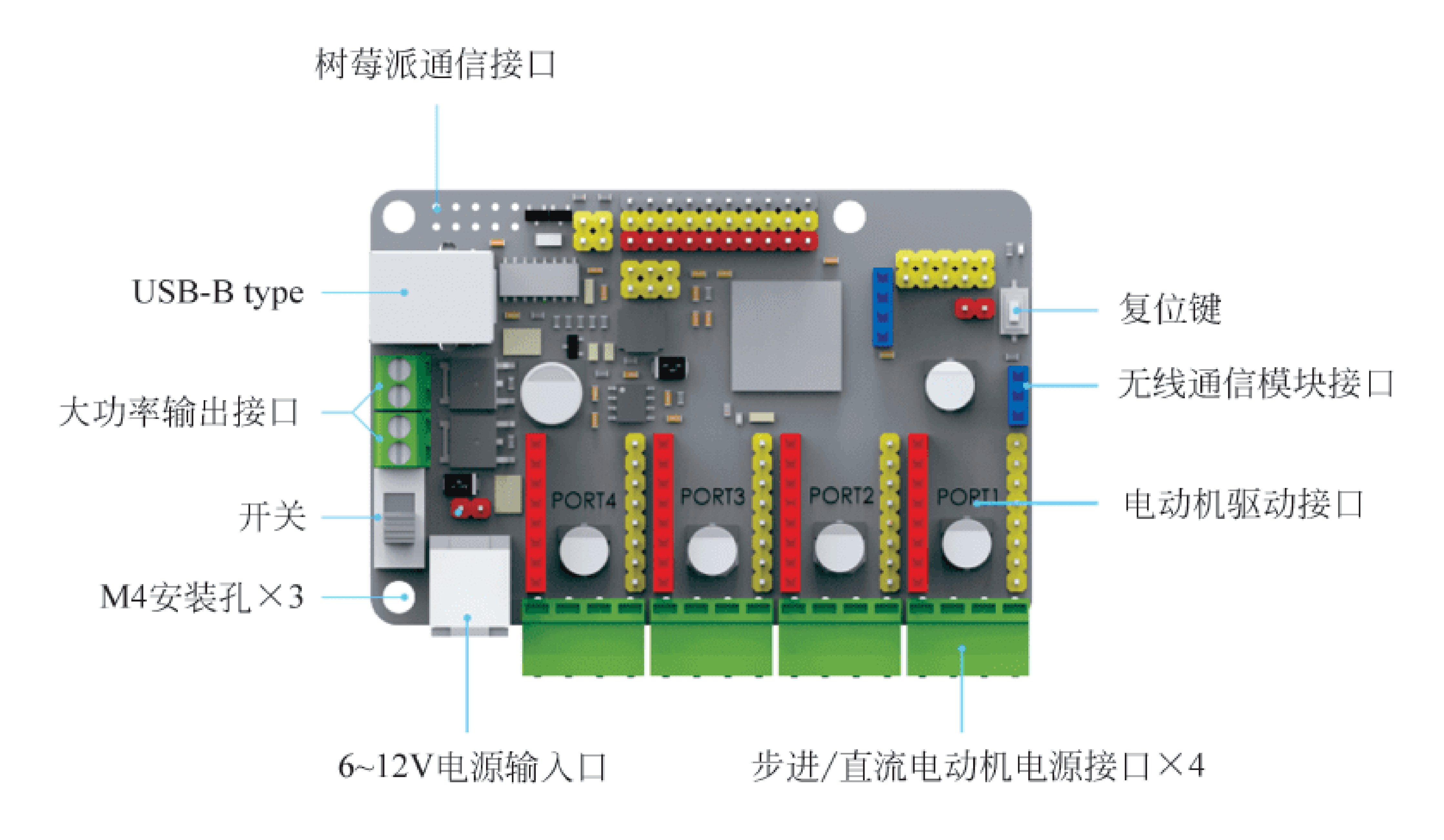


图2–3　MegaPi接口

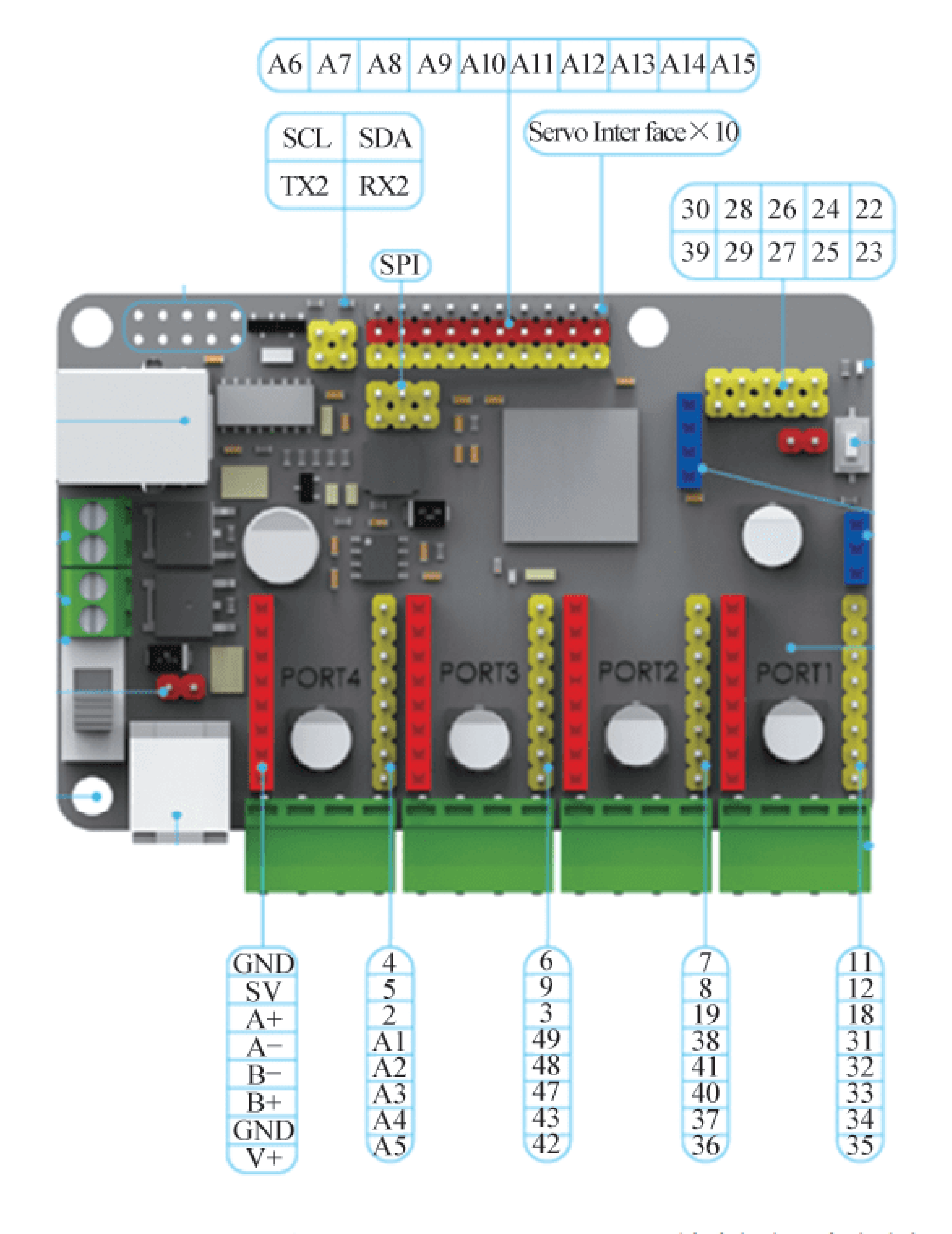


图2–4　MegaPi和Arduino Mega 2560的针脚对应关系

MegaPi 主控板上面不同的颜色代表不同的功能，如下所述。

（1）红色代表电源输出或者电动机输出。

（2）黄色代表 I/O 端口。

（3）蓝色代表无线通信接口。

（4）黑色代表电源接地。

（5）绿色代表电源输出或电动机输出。

2.1.3　MegaPi 丰富的端口

MegapPi 具有 Arduino Mega 2560 的所有端口，只是针对项目需求，MegaPi 专门进行了封装。

MegaPi 扩展板采用 RJ25 接口。RJ25 适配器将标准的 RJ25 接口转换为 6 个引脚（分别为 VCC、GND、S1、S2、SDA、SCL），方便从 Makeblock 接口引出来以兼容其他厂商的电子模块。Makeblock 公司提供了大量的基于 RJ25 接口的电子模块，通过模块上的颜色就能知道可以接入的端口（Port）。图 2-5 列出了 MegaPi 的端口定义，图 2-6 列出了部分 MegaPi 兼容的电子模块。

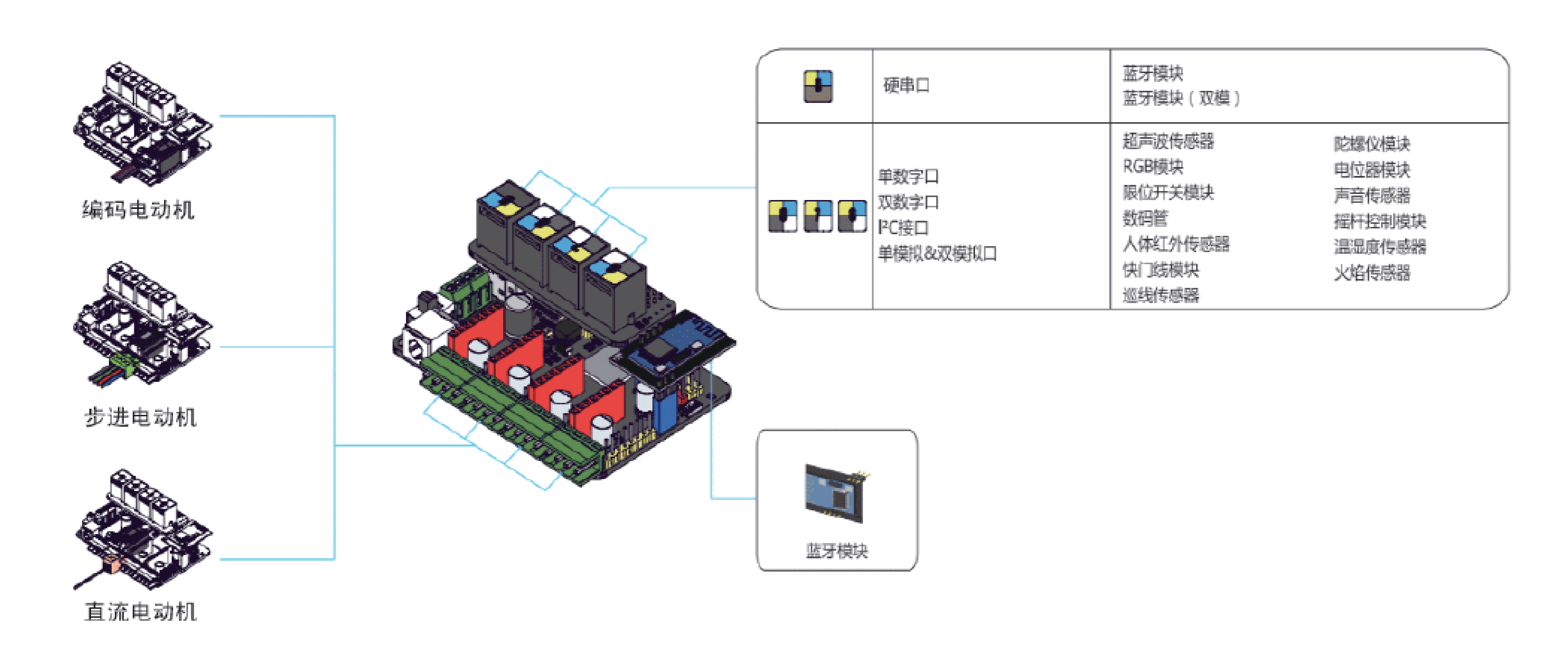


图2-5　MegaPi的端口定义

MegaPi 的 RJ25 插板提供了由 5 种颜色搭配标记的 4 个 RJ25 接口，功能如图 2-5 所示。

（1）无线接口：接蓝牙模块或者 2.4G 无线模块。

（2）Port1~Port4：接电动机驱动，驱动 DC 电动机、步进电动机和编码电动机。

（3）Port5：接 RJ45 通信模块、蓝牙或者 Wi-Fi 模块。

（4）Port6~Port8：数模 RJ25 接口，接传感器和输入 / 输出模块。

图 2-6 是部分 MegaPi 兼容的电子模块。





图2-6　部分MegaPi兼容的电子模块



2.1.4　MegaPi 编程环境搭建

能够为 MegaPi 编程的环境有很多，这里主要介绍 Python 的编程环境。

由于 Python 是解释型语言，所以上位机要通过串口和 MegaPi 通信，一条一条解释执行 Python 命令。

MegaPi 端需要安装新的固件，方法如下。

打开 Arduino IDE，选择菜单命令 File → Examples，再选择 firmata → standardFirmata 程序。

编译和上传相应的固件，上传成功后，MegaPi 的准备工作就已经完成了。

Python 端通过 1.1.4 小节中讲到的关于 Python 模块安装的内容，安装 Python 库文件 pyfirmata。命令如下：

```
pip install pyfirmata
pip install serial
```

目前，支持 Arduino 的 Python 扩展库非常多，如果有兴趣，还可以尝试下 Pyserial 和 Arduino 等扩展库。

以上工作完成后，我们就可以在 Windows 环境中，利用 Python 编程控制 MegaPi 了。

2.2 Python控制MegaPi

2.2.1 Blink

学过 Arduino 的人都知道 Blink 程序示例让板载的 LED 闪烁。我们现在就让 MegaPi 主控板上的 LED 也闪烁起来。

```
from pyfirmata import Arduino, util
import time
board = Arduino('COM7')              #MegaPi 所在串口是 COM7
while 1:
    board.digital[13].write(0)       # 向端口 13 写入 0
    time.sleep(1)                    # 休眠 1s
    board.digital[13].write(1)       # 向端口 13 写入 1
    time.sleep(1)
```

运行程序，可以看到，主控板上的蓝色 LED 灯间隔 1s 闪烁起来。

2.2.2 控制舵机

连接舵机到 13 号数字端口，输入如下程序：

```
from pyfirmata import Arduino, util
import time
board = Arduino('COM7')              #Arduino 所在串口是 COM7
    while 1:
        board.servo_config(13,0,255,20)
        print("ceshi")
        time.sleep(1)
        board.servo_config(13, 0, 255, 255)
        time.sleep(1)
```

本书的重点在于树莓派与 MegaPi 结合，PC 端用于控制 MegaPi 的 Python 编程不是重点，在此不再多做介绍，有兴趣的读者可以自行深入研究。



第 3 章
树莓派基础

3.1 树莓派简介

3.1.1 树莓派的应用场合

树莓派是为学习计算机编程而设计的，其系统基于 Linux。它由注册于英国的慈善组织 Raspberry Pi 基金会开发。它的外形只有信用卡大小，却具有计算机的所有基本功能。别看其外表“娇小”，内“心”却很强大，视频、音频等功能皆有，可谓“麻雀虽小，五脏俱全”。只须接通电视机和键盘，就能执行如电子表格、文字处理、玩游戏、播放高清视频等诸多功能。

与 PC 和笔记本电脑相比，树莓派的处理能力要差很多，不适合应用于对处理能力要求较高的场合，但相比于 Arduino、STM32 等较为流行的单片机系统，它的处理能力高出一大截。普通计算机主板是依靠硬盘存储数据的，而树莓派使用 SD 卡作为“硬盘”，也可以外接 USB 硬盘。

树莓派价格低廉，这意味着其用途更加广泛，将其打造成卓越的多媒体中心也是一个不错的选择。利用树莓派可以播放视频，甚至可以通过电视机的 USB 接口供电。

树莓派适合的应用场合如下。

（1）作为一个低能耗的 Linux 家用服务器，用于运行硬件性能可以满足的软件，可提供各种服务（如网络相关、文件相关、视频音频相关）。

（2）连接硬件用来做数据采集、监控、分析、发布等事情。

（3）作为类似小车、飞行器、机器人、智能家居等智能设备的控制中心。

（4）作为一个计算机的轻量级替代物，用于一些简单的场景，或者用来作软件开发启蒙，适合预算不足以购买计算机的用户（主要是孩子）。这个是设计树莓派的最初目的。

（5）用于青少年的编程学习。

（6）用于搭建原型产品。



3.1.2　树莓派 3

本书主要介绍树莓派 3 的应用。

树莓派 3 实物如图 3-1 所示。

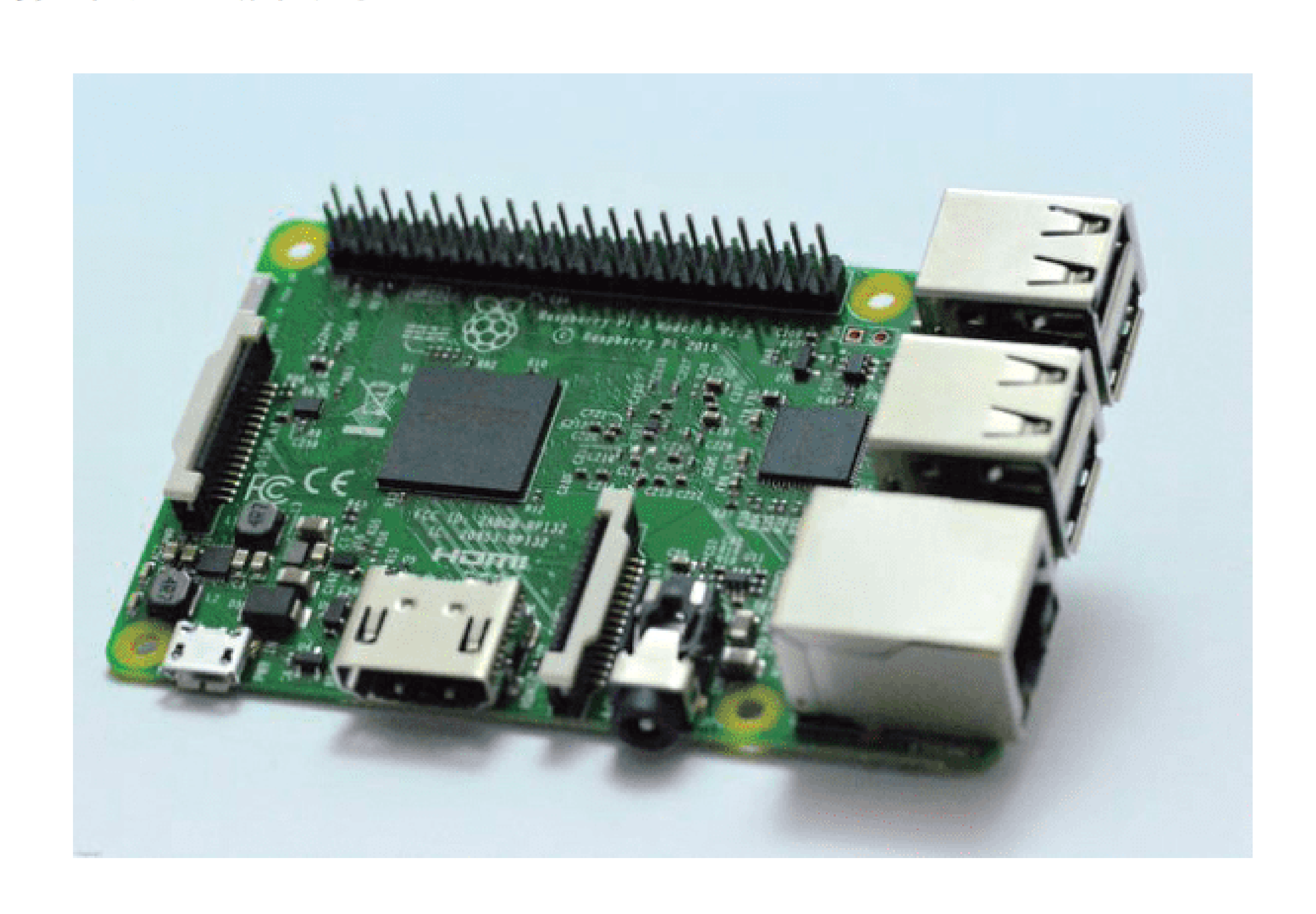


图3-1　树莓派 3 实物

树莓派 3 是第三代树莓派，硬件资源如下。

（1）CPU 为 4 核 1.2GHz 博通 BCM2837，64 位处理器，1GB RAM。

（2）板载 BCM43438 无线 Wi-Fi 模块和低功耗蓝牙模块。

（3）40 个扩展 GPIO 口，4 路立体声输出和复合视频端口。

（4）全尺寸 HDMI 接口，Micro SD 卡插槽。

（5）CSI 摄像头接口，可以连接树莓派摄像头。

（6）DSI 显示屏接口，可以连接树莓派专用显示屏。

3.1.3　树莓派 GPIO 与引脚编号

树莓派的 GPIO 引脚是数字引脚，可以将它的输出设为高或低，或者通过它读取输入的高低电平。如果想读取模拟输入设备的值，还需要使用 ADC（模数转换器）芯片。

树莓派 GPIO 引脚编号的方式有两种：一种是 GPIO 编号；另一种是板上的自然编号。GPIO 编号是 Broadcom 提供的一种编号规则，它和 Broadcom 片上系统中的信道编号相对应，这些编号看起来没有什么规律，也没有什么好的办法记住它们；自然编号是根据板子上引

脚的位置进行编号，自下而上，从左到右，依次进行编号。表 3-1 列出了树莓派两种引脚编号的对照表。

表 3–1　树莓派引脚编号的对照表

wiringPi 编码	BCM 编码	功能名	物理引脚 BOARD编码		功能名	BCM 编码	wiringPi 编码
		3.3V	1	2	5V		
8	2	SDA.1	3	4	5V		
9	3	SCL.1	5	6	GND		
7	4	GPIO.7	7	8	TXD	14	15
		GND	9	10	RXD	15	16
0	17	GPIO.0	11	12	GPIO.1	18	1
2	27	GPIO.2	13	14	GND		
3	22	GPIO.3	15	16	GPIO.4	23	4
		3.3V	17	18	GPIO.5	24	5
12	10	MOSI	19	20	GND		
13	9	MISO	21	22	GPIO.6	25	6
14	11	SCLK	23	24	CE0	8	10
		GND	25	26	CE1	7	11
30	0	SDA.0	27	28	SCL.0	1	31
21	5	GPIO.21	29	30	GND		
22	6	GPIO.22	31	32	GPIO.26	12	26
23	13	GPIO.23	33	34	GND		
24	19	GPIO.24	35	36	GPIO.27	16	27
25	26	GPIO.25	37	38	GPIO.28	20	28
		GND	39	40	GPIO.29	21	29

Python 既可以使用 GPIO 编号，也可以选择使用自然编号。

3.1.4　树莓派操作系统 Raspbian

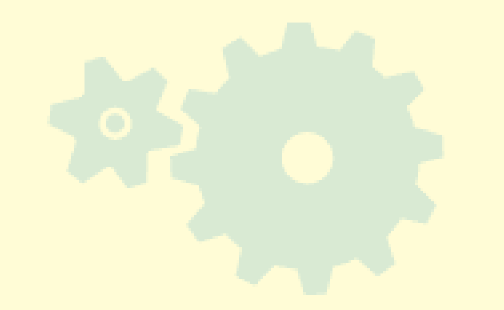
Raspbian 操作系统是单纯的 ARM 版的 Linux 系统，它基于 Debian，也是图形化的操作系统。Raspbian 操作系统预安装了丰富的软件，非常适合以教育为目的的编程学习，包含 Python、Scratch、Sonic Pi、Java、Mathematica 等软件。

Raspbian 下载地址：https://www.raspberrypi.org/downloads/raspbian/。

Raspbian 桌面操作系统镜像文件是一个 zip 格式、大小为 4GB 的文件，在一些平台上，如果解压工具比较老，可能无法解压该文件，此时可以使用官方推荐的工具。

（1）Windows 平台：7-zip。下载地址为 http://www.7-zip.org/download.html。

（2）Linux 平台：unzip。ubuntu 安装命令为 sudo apt-get install unzip。



3.1.5 制作 SD 卡启动盘

Etcher 是一个图形界面的 SD 卡烧写工具，对于大多数用户来说，这是一个非常好的选择。Etcher 也支持 zip 文件直接烧写，不需要解压。

Etcher 的下载地址为 https://etcher.io/。

将 SD 卡插入读卡器，打开 Etcher，选择上面下载的 .zip 文件，烧写到 SD 卡里面。单击 Flash！按钮，开始烧写树莓派操作系统到 SD 卡里面。

3.2 树莓派开发环境

3.2.1 树莓派启动

将制作好的 SD 卡插入树莓派的 SD 卡插槽，插入网线、键盘和鼠标，然后通电。系统启动后如图 3-2 所示。

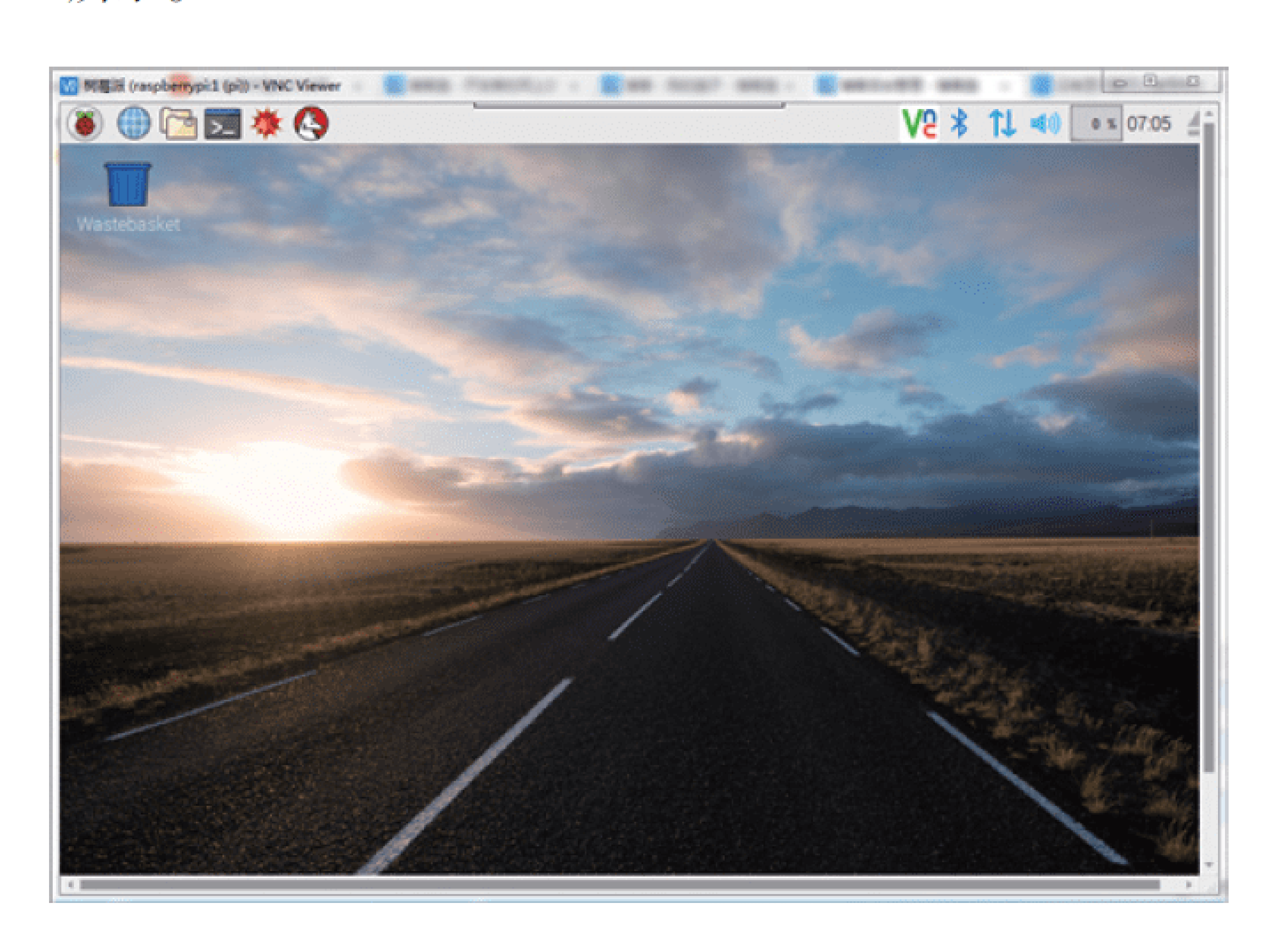


图3–2 树莓派操作系统Raspbian启动成功

3.2.2 搭建树莓派开发环境

可以通过 SSH 远程网络连接，在其他计算机或者设备上访问树莓派的命令行，以实现



控制树莓派。在这种方式里树莓派作为一个远程设备，我们的计算机作为一个连接客户端。这种方式只能访问树莓派的命令行终端，要想访问桌面，必须通过 VNC。接下来的章节会详细讲解。

确保树莓派物理连接正确并且通过网线插入路由器里面，打开一个叫 Terminal 的终端窗口，并输入如下命令：

```
$ ifconfig 或者 $ hostname -I
```

这样就可以得到树莓派的 IP 地址，例如我的树莓派显示为 192.168.0.102。

3.2.3　使能 SSH

SSH 为 Secure Shell 的缩写，由 IETF 的网络小组（Network Working Group）制定。SSH 为建立在应用层基础上的安全协议。SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

Raspbian 操作系统自从 2016 年 11 月发行后，默认禁用 SSH 服务器，因此我们需要打开它。打开方式如下。

单击左上角树莓派图标，启动应用选择菜单，选择 Preferences 菜单命令，在弹出的菜单栏中选择 Raspberry Pi Configuration，如图 3-3 所示。

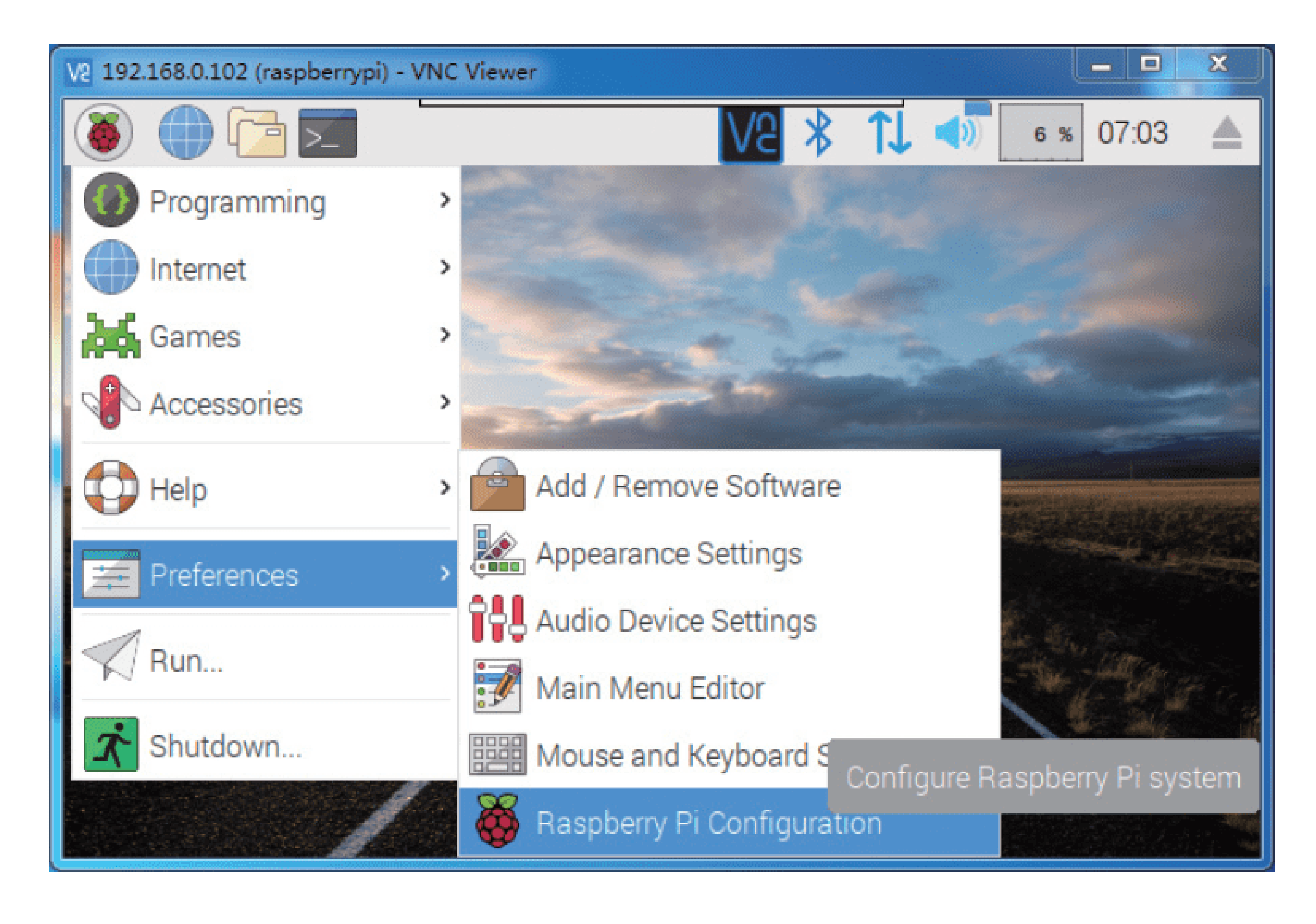


图3–3　使能SSH设置

单击启动树莓派设置，选择使能 SSH，如图 3-4 所示。

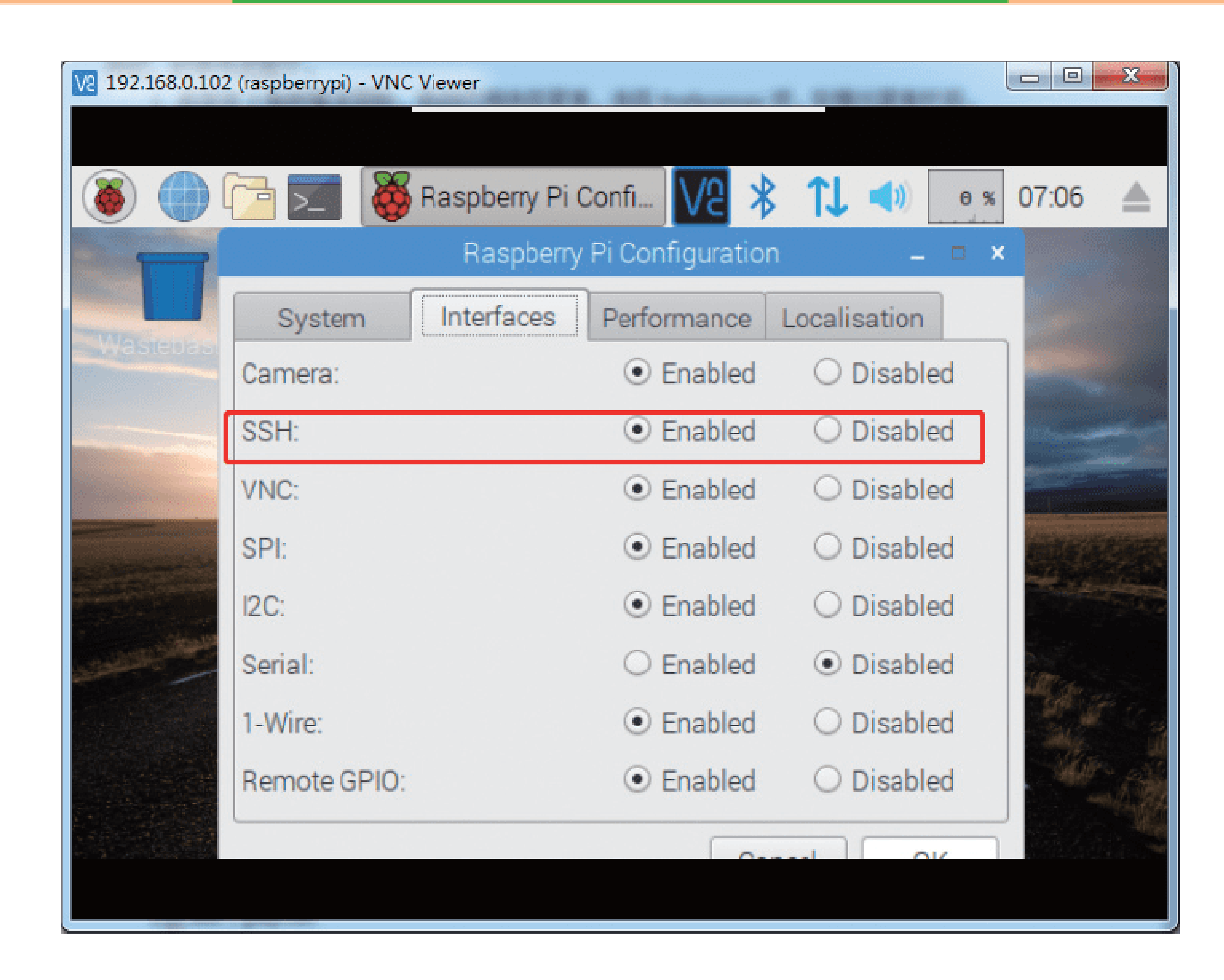


图3–4 使能SSH设置

在一台 Windows 操作系统的计算机上面，从官网 https://www.putty.org/ 下载软件 PuTTY，打开 PuTTY 程序，如图 3-5 所示。

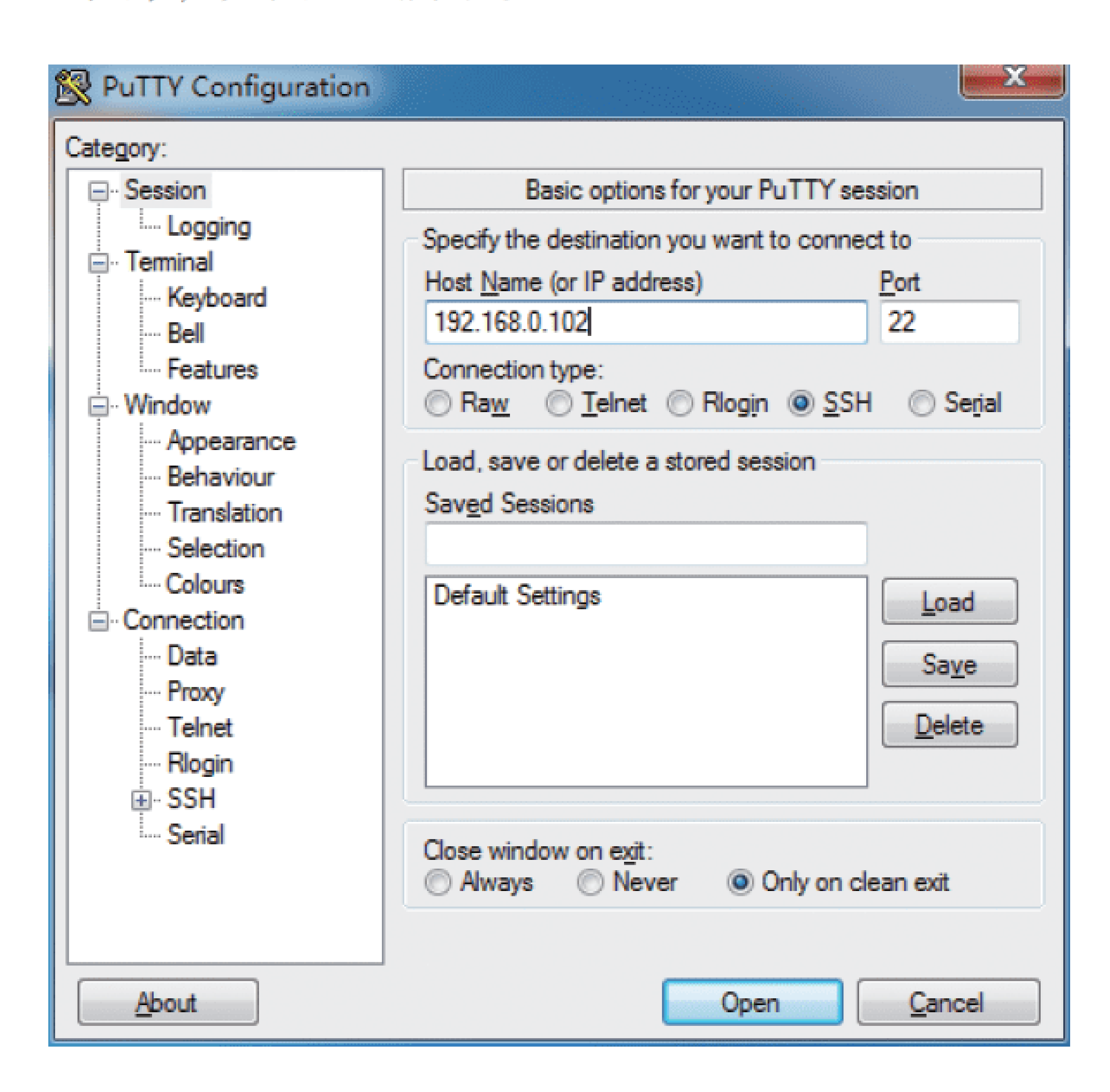


图3–5 PuTTY设置

输入上面得到的 IP 地址，单击 Open 按钮，就会跳转到树莓派的远程登录界面，如图 3-6 所示。



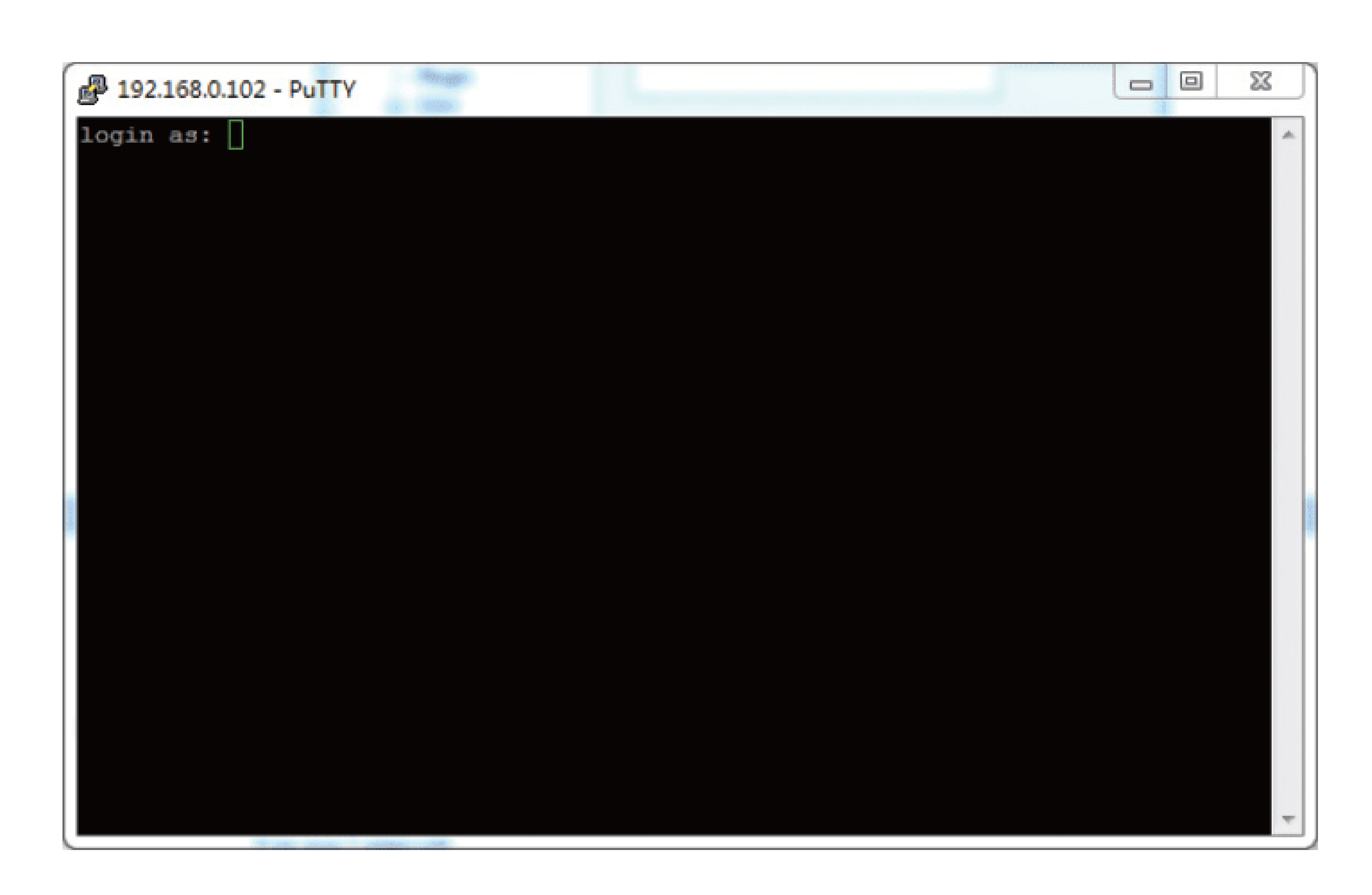


图3–6　远程登录

登录的用户名为 pi；登录的初始密码为 raspberrypi。成功登录后如图 3-7 所示。

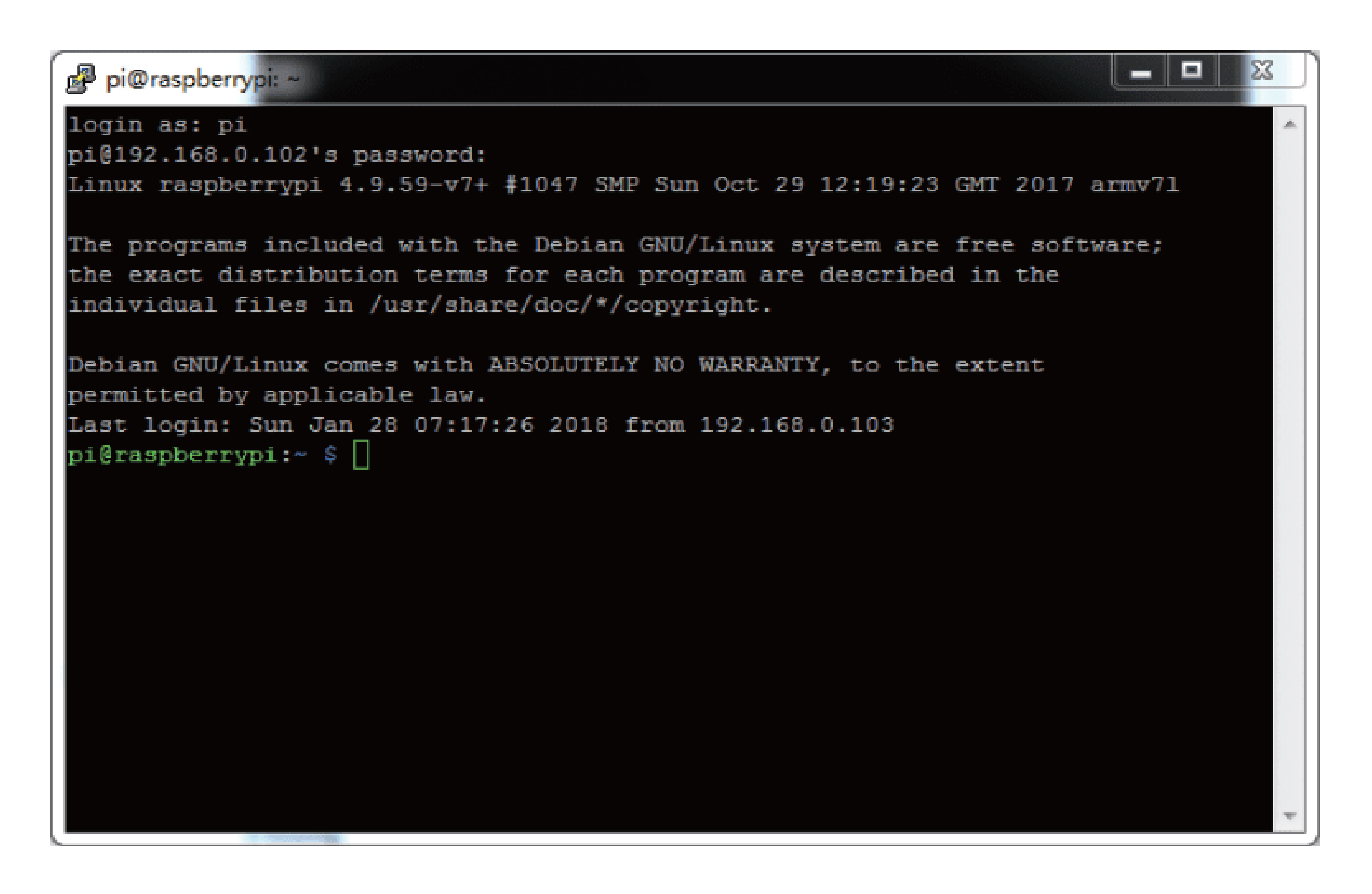


图3–7　远程登录成功

3.2.4　使能 VNC

VNC 是 Virtual Network Computing 的缩写，它是一种桌面共享系统，允许访问者通过网络访问树莓派的桌面系统，使能方式跟使能 SSH 一样，首先在树莓派上面安装 VNC 服务器：

```
$ sudo apt-get install realvnc-vnc-server realvnc-vnc-viewer
```

安装完 VNC 后，打开树莓派的设置，使能 VNC，跟打开使能 SSH 一样。这样就可以通过客户端软件访问树莓派的桌面系统，如图 3-8 所示。

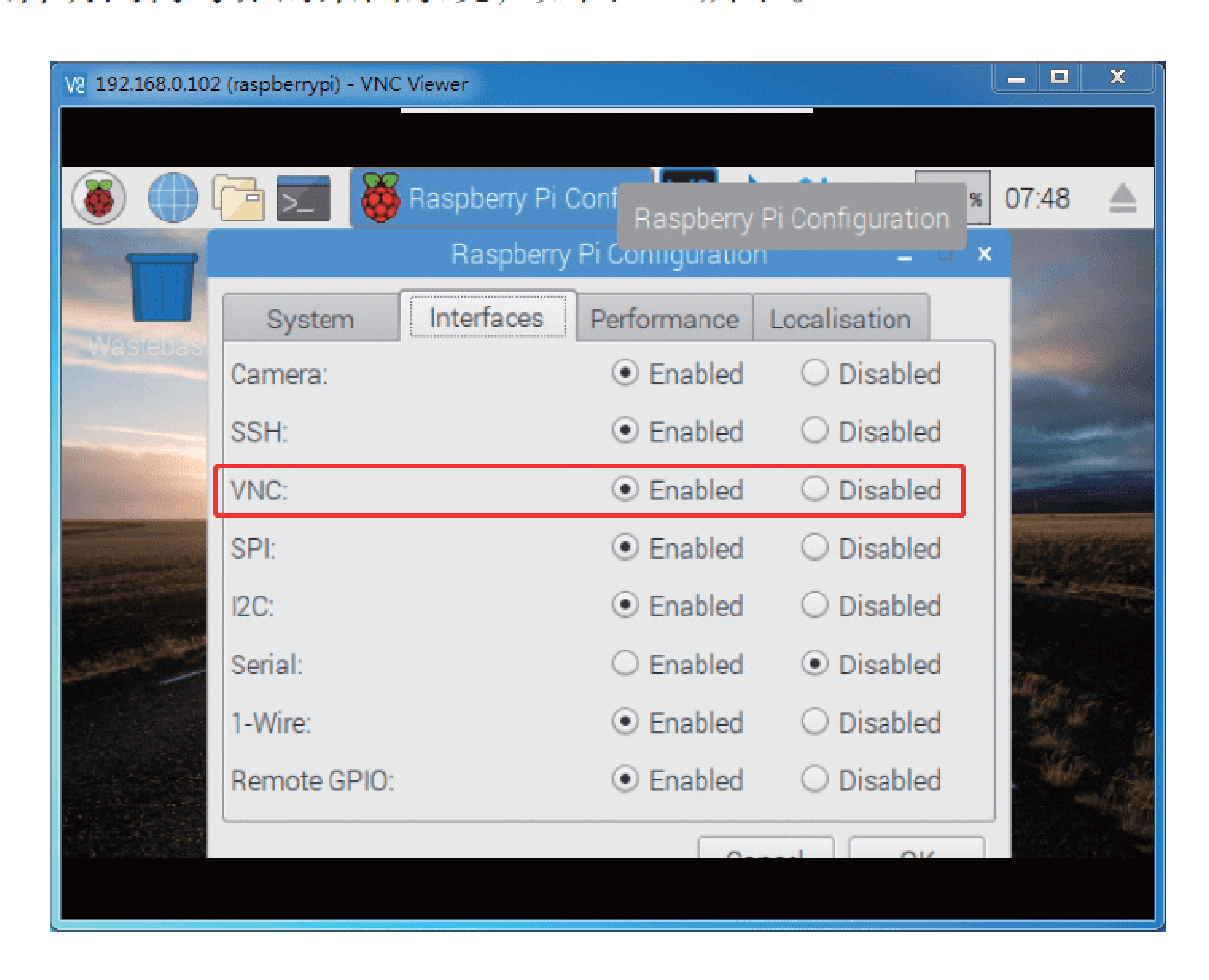


图3–8　使能VNC

下载对应操作系统版本的 VNC 客户端连接软件，地址如下：

https://www.realvnc.com/en/connect/download/vnc/

下载完成后，双击打开，如图 3-9 所示。

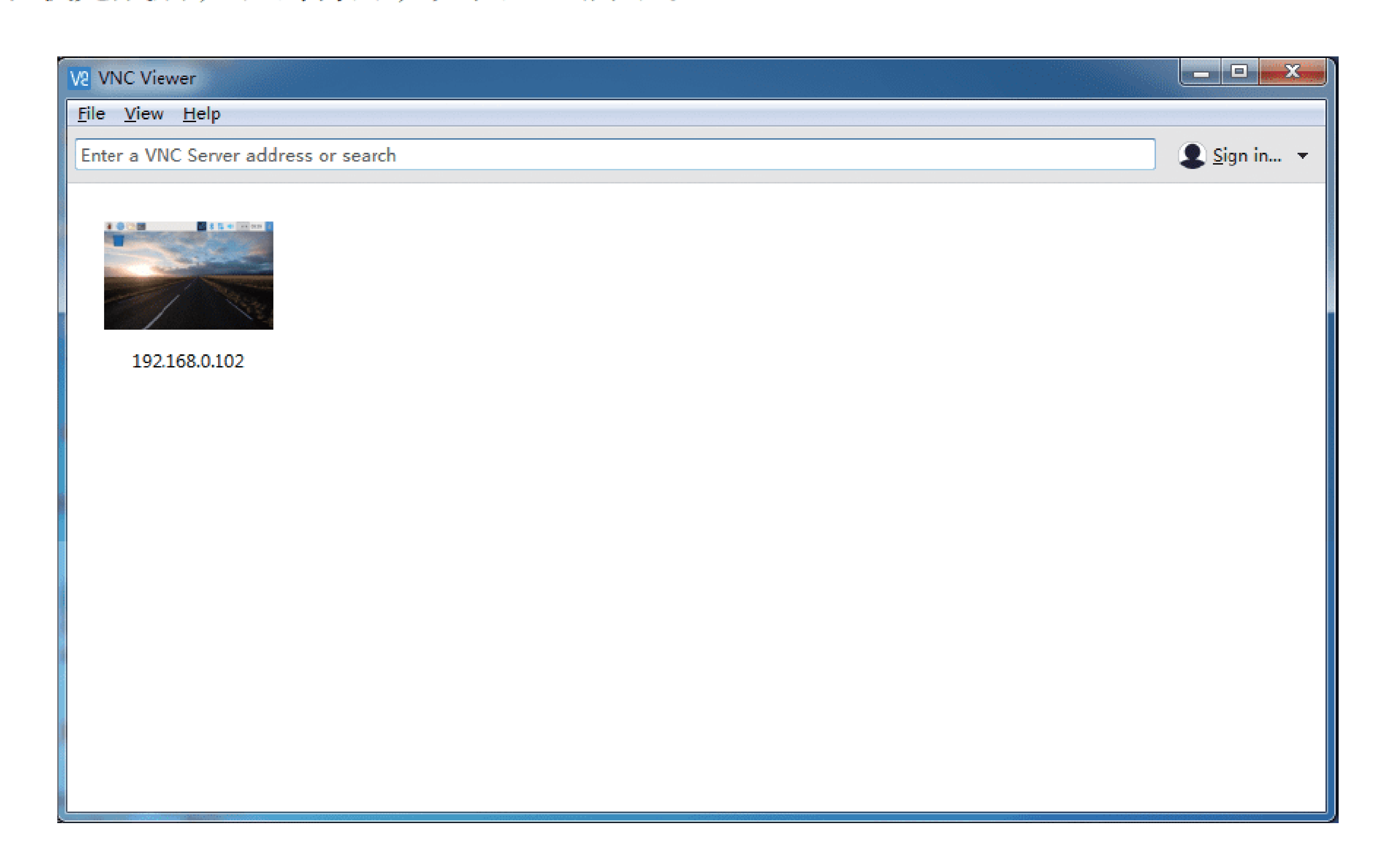


图3–9　VNC窗口



在输入框内输入树莓派的 IP 地址，就会弹出登录窗口，输入用户名和密码，用户名为 pi，登录密码为 raspberrypi，如图 3-10 所示。

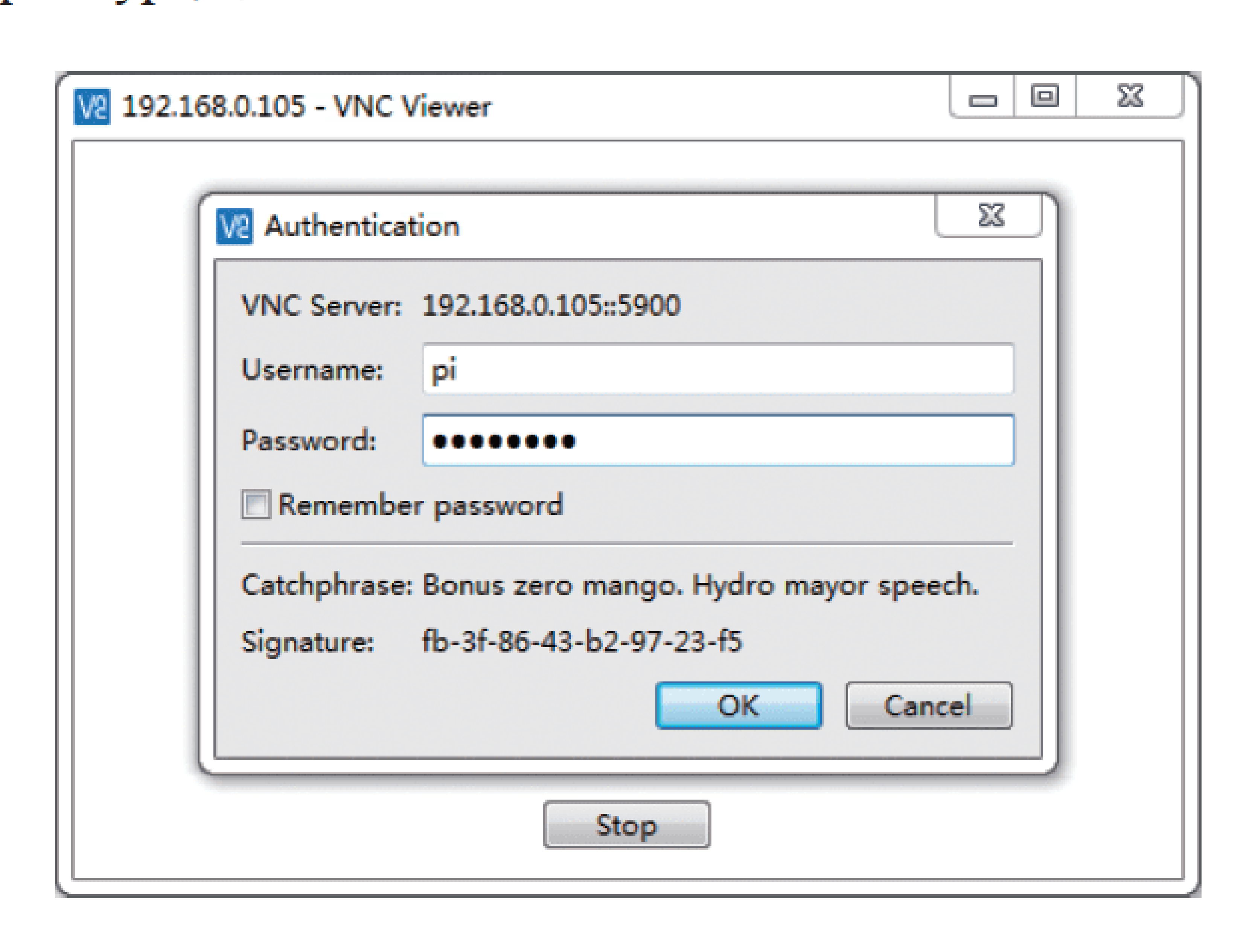


图3-10　登录VNC

单击 OK 按钮，成功登录界面如图 3-11 所示。

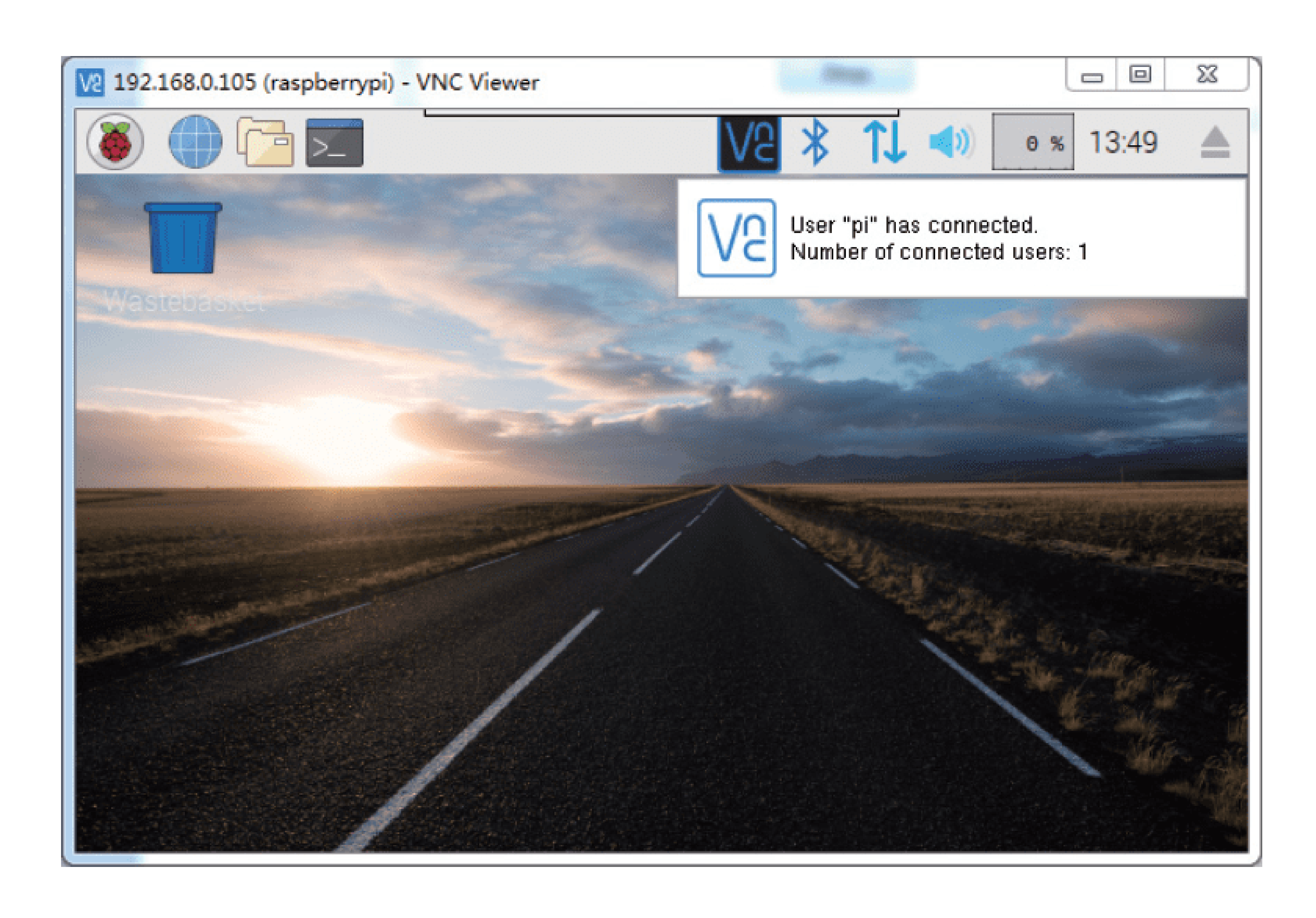


图3-11　Windows远程登录树莓派

可以看到，我们已经成功访问了树莓派的操作系统桌面，就像在自己的计算机上一样。这样就可以在 Windows 系统上远程操作树莓派了。

安装 SSH 是为了简化无图形界面的开发，安装 VNC 则是为了后面的图像识别等图像程序开发更便捷。



3.2.5　中文显示和安装拼音输入法

树莓派默认安装系统是英文版，还需要为系统安装中文输入法才可以使用中文。

打开命令窗口，输入如下命令：

```
sudo apt-get install ttf-wqy-microhei scim-pinyin
```

从 Preferences 打开 Raspberry Pi Configuration 窗口，单击 Set Locale 按钮，在弹出的对话框中设置 Language 为 zh(Chinese)，Country 为 CN(P.R.of China)，Character Set 为 UTF-8，如图 3-12 所示。然后重启树莓派。

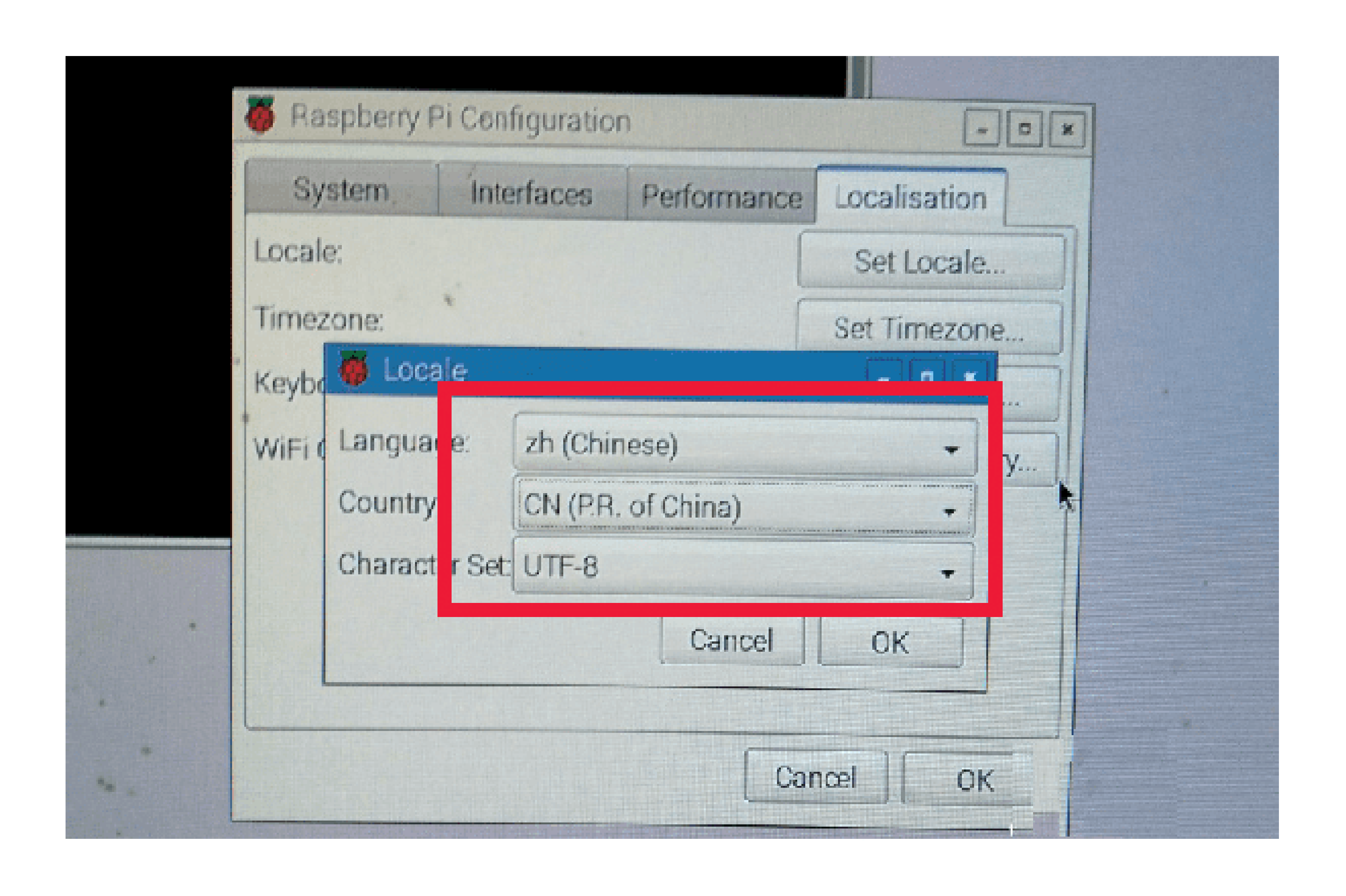


图3-12　树莓派中文设置

3.3　Python控制树莓派GPIO

使用 Python 控制树莓派，需要安装树莓派官方资料中推荐的扩展库，名称为 raspberry-gpio-python。Python GPIO 是一个小型的 Python 库，可以帮助用户完成 raspberry 相关的 I/O 端口操作。但是 Python GPIO 库目前还没有支持 SPI、I^2C 或者 1-wire 等总线接口。

树莓派官方还提供了完整的 Python GPIO 库 gpiozero，并且可以随系统一起安装，然后直接使用，只需要导入库就行了。gpiozero 库使用 Broadcom（BCM）引脚编号作为 GPIO 引脚，而不是物理（BOARD）编号。与 RPi.GPIO 库不同，这是不可配置的，使用时一定要加以注意。



3.3.1 配置环境

启动树莓派，在命令行窗口输入以下命令，安装 Python 扩展库：

```
sudo apt-get install python-dev
sudo apt-get install python-rpi.gpio
sudo apt-get install python-serial
```

3.3.2 Blink

电路连接示意如图 3-13 所示。

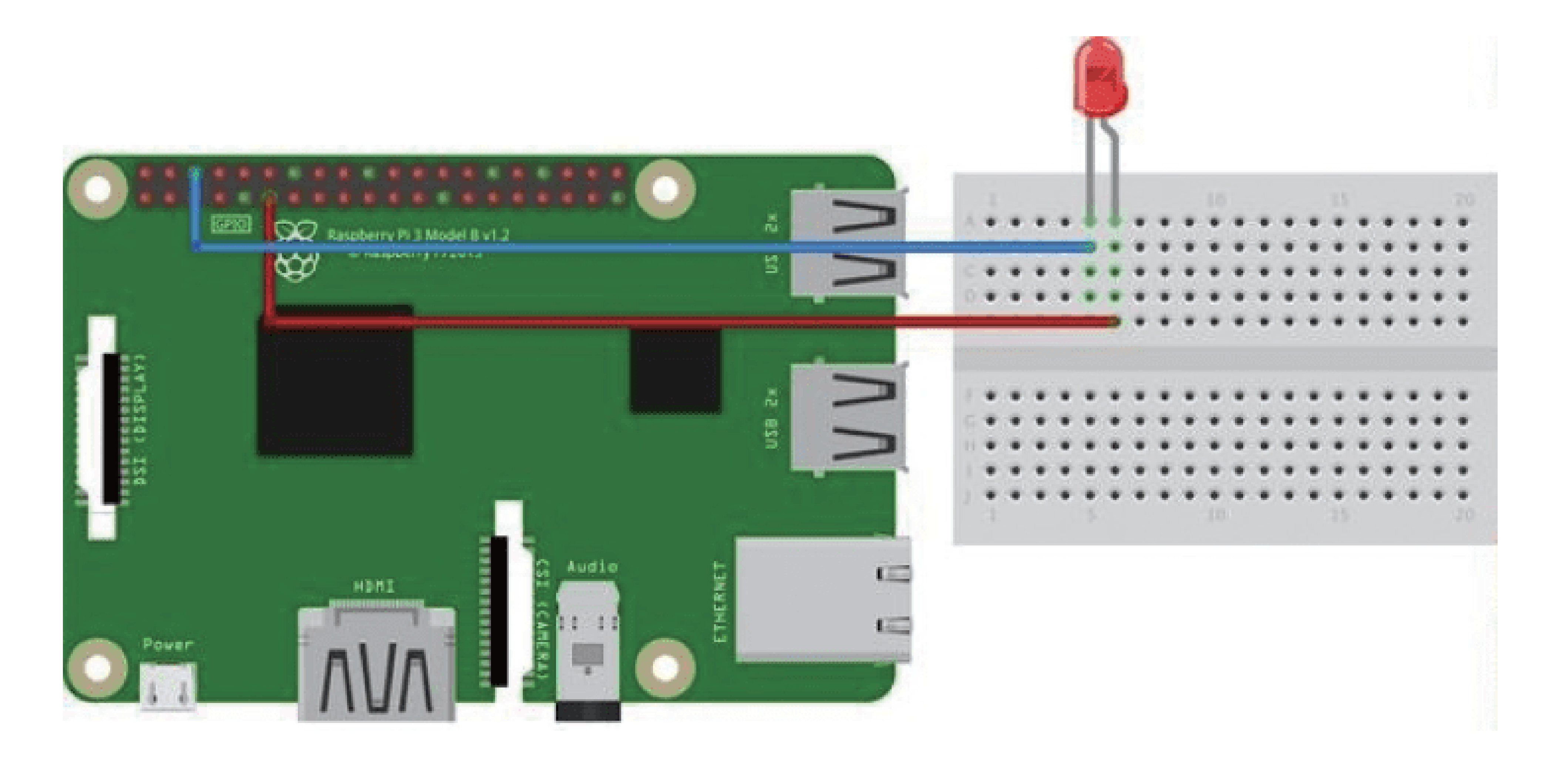


图3–13 电路连接示意图1

（1）基于 RPI.GPIO 扩展库程序示例。

打开 Python，建立新文件，输入如下代码：

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)        #BOARD 编号方式，基于插座引脚编号
GPIO.setup(11, GPIO.OUT)        # 输出模式
while True:
   GPIO.output(11, GPIO.HIGH)
   ptime.sleep(1)
   GPIO.output(11, GPIO.LOW)
   time.sleep(1)
```

运行程序，就会看到 11 号针脚的 LED 灯闪烁。

（2）基于 gpiozero 官方扩展库程序示例。

打开 Python，建立新文件，输入如下代码：

```
from gpiozero import LED        # 从 gpiozero 库中导入 LED 函数
from time import sleep
red = LED(17)                   # LED 灯的正极接 GPIO17
while True:
    red.on()                    # 开灯
    sleep(1)
    red.off()                   # 关灯
    sleep(1)
```

运行程序，就会看到同样的 LED 灯闪烁。

上面的程序还可以改写成更简明的方式：

```
from gpiozero import LED
from signal import pause
red = LED(17)                   # LED 灯的正极接 GPIO17
red.blink()                     # 闪烁
pause()
```

3.3.3　改变 LED 灯的亮度与呼吸灯

电路连接示意如图 3-13 所示。

任何常规的 LED 灯都可以使用 PWM（脉冲宽度调制）设置其亮度值。在 gpiozero 中，可以使用 PWMLED 来实现，PWMLED 的值从 0 到 1。

```
from gpiozero import PWMLED
from time import sleep
led = PWMLED(17)
while True:
    led.value = 0               # 灭
    sleep(1)
    led.value = 0.5             # 半亮
    sleep(1)
    led.value = 1               # 全亮
    sleep(1)
```

类似于连续闪烁，PWMLED 可以控制占空比，实现呼吸灯的效果（连续淡入和淡出）:

```
from gpiozero import PWMLED
from signal import pause
led = PWMLED(17)
led.pulse()                     # 呼吸灯的效果
pause()
```

运行程序，可以看到 LED 灯的亮度连续淡入和淡出。



3.3.4　使用一个按钮控制 LED 灯

电路连接示意如图 3-14 所示。

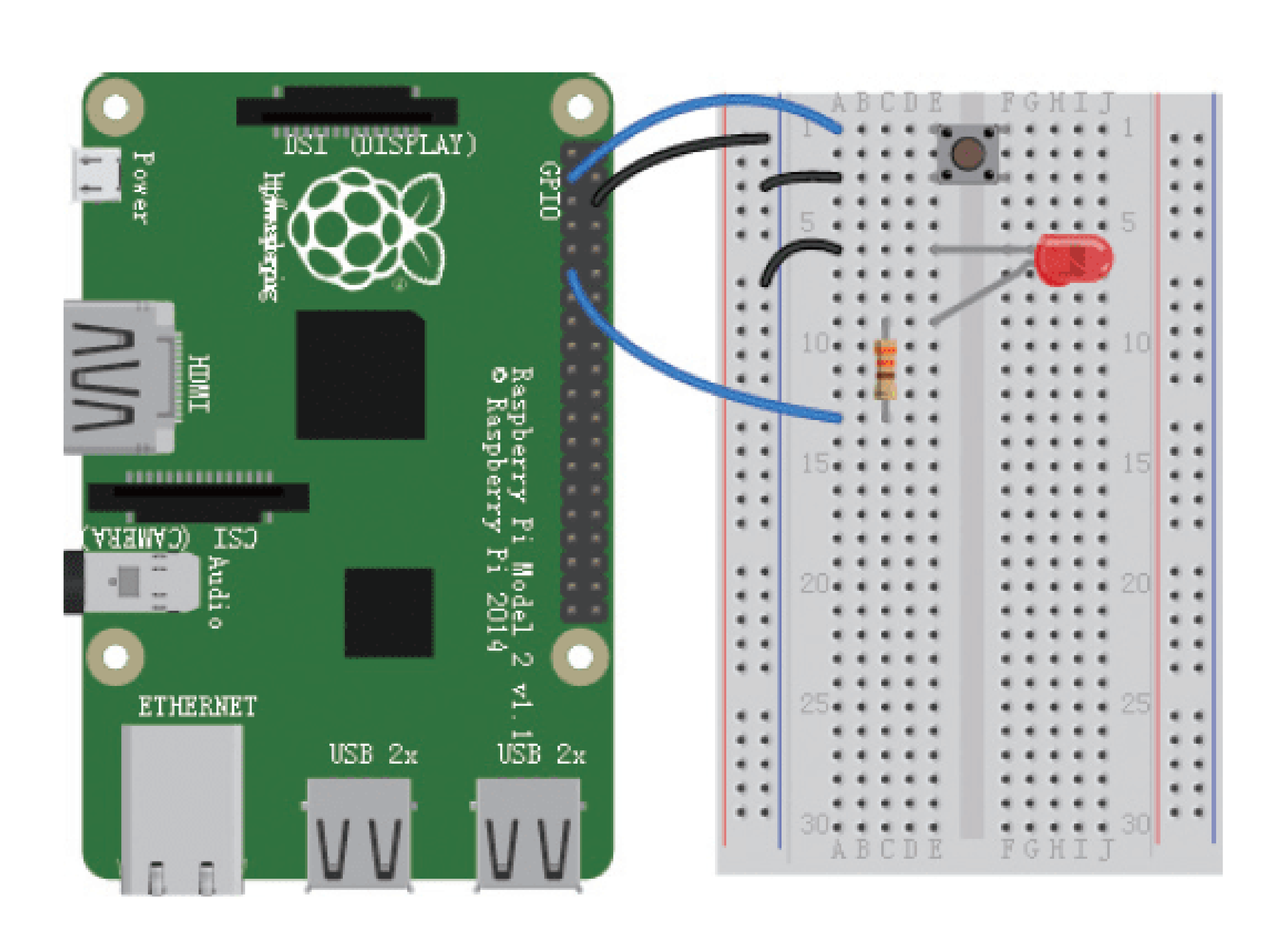


图3-14　电路连接示意图3

按下按钮时打开 LED 灯：

```
from gpiozero import LED, Button
from signal import pause
led = LED(17)                     #定义一个 LED 灯
button = Button(2)                #定义一个 Button
button.when_pressed = led.on      #开灯
button.when_released = led.off    #关灯
pause()
```

或者：

```
from gpiozero import LED, Button
from signal import pause
led = LED(17)                     #定义一个 LED 灯
button = Button(2)                #定义一个 Button
led.source = button.values
pause()
```

3.3.5　控制蜂鸣器发声

蜂鸣器有两种：有源（Active）和无源（Passive）。有源蜂鸣器只要通电就会发出固定频率的声音；无源蜂鸣器可以发出不同频率的声音。这里用 Python 让有源蜂鸣器发声。

Pi.GPIO 库使用 output 方法输出高低电平并不能有效地控制蜂鸣器的发声，因为高低

电平都会响，最终勉强使用 input 和 output 方法转换接口的 I/O 状态实现嘀嘀的间隔音。使用 gpiozero 库就简单多了。

蜂鸣器正极接 GP17 针脚，负极接 GND。程序如下。

```
from gpiozero import Buzzer
from time import sleep
bz = Buzzer(17)
while 1：
     bz.on()
     sleep(1)
bz.on()
sleep(1)
```

或者：

```
from gpiozero import Buzzer
from time import sleep
bz = Buzzer(17)
while 1：
bz.beep()
```

运行程序，蜂鸣器发出“嘟嘟”的声音。

3.3.6　相机图像的预览

树莓派摄像头模块如图 3-15 所示。摄像头与树莓派通过一条 15 芯的排线连接，如图 3-16 所示。

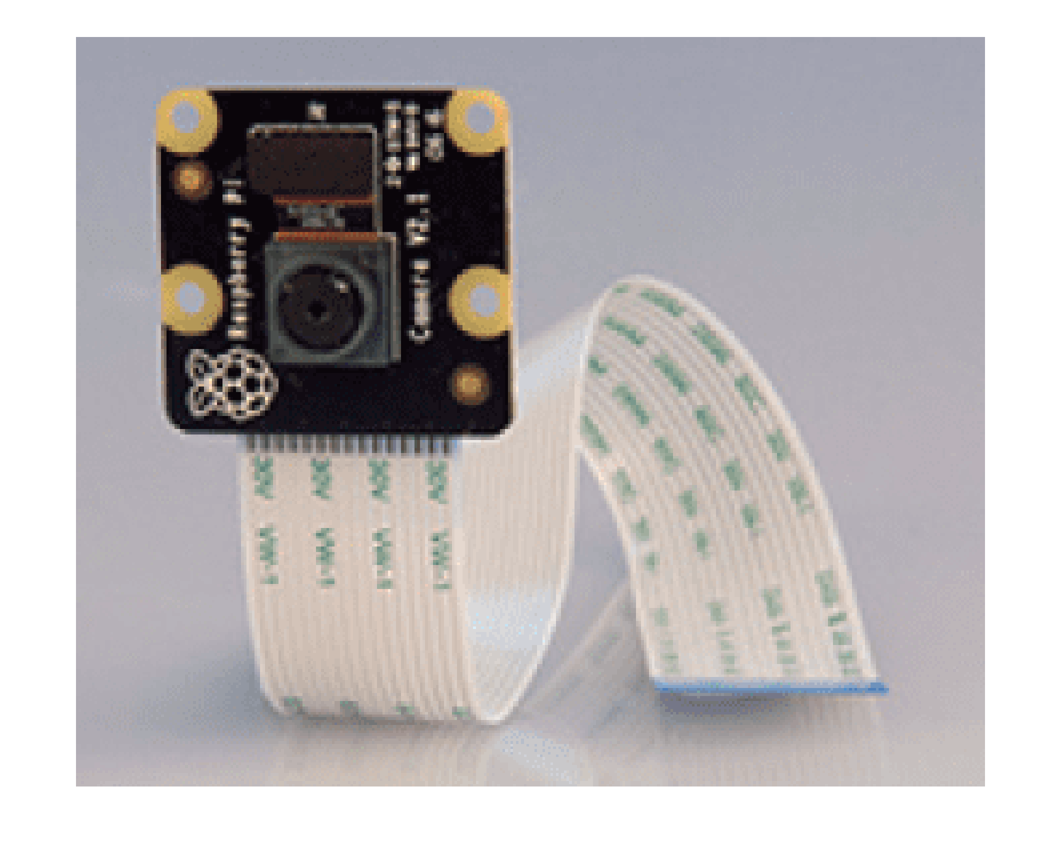


图3-15　树莓派摄像头模块

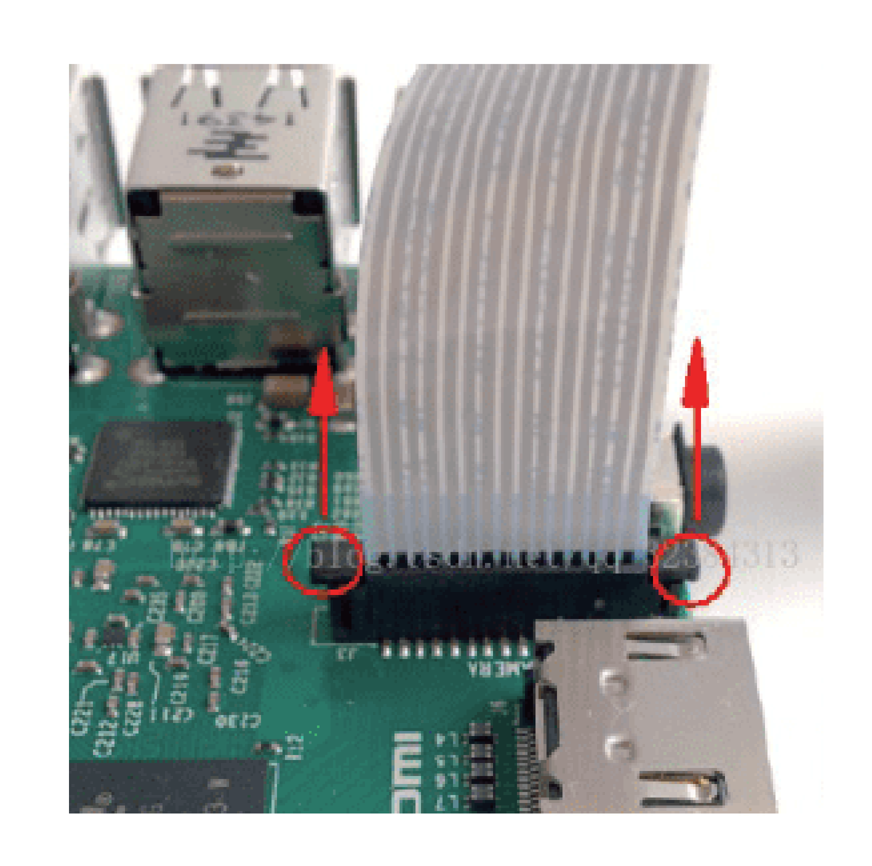


图3-16　摄像头与树莓派的连接

打开 Python，建立新文件，输入如下代码：

```
from picamera import PiCamera
from time import sleep
```




```
camera = PiCamera()
camera.start_preview()
sleep(10)
camera.stop_preview()
```

运行程序，就会看到相机的预览。

如果图像是上下颠倒的，可以用一下代码实现翻转，代码如下：

```
camera.rotation = 180
camera.start_preview()
sleep(10)
camera.stop_preview()
```

可以设置旋转的度数为 90°、180° 或 270°，也可以设置为 0。

通过改变 alpha 的值，还可以修改摄像头拍摄图像的透明度。

```
from picamera import PiCamera
from time
import sleep
camera = PiCamera()
camera.start_preview(alpha=200)
sleep(10)
camera.stop_preview()
```

alpha 的取值范围为 0~255。

3.3.7 拍摄照片

修改代码：减少 sleep 并添加 camera.capture( ) 一行。

```
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

捕获图片前，至少要给传感器 2s 时间感光。运行程序 5s 后，相机模块会拍摄一张名为 image.jpg 的照片并保存在桌面上。

增加一个循环就能实现连拍，每隔 5s 拍一张，拍完后预览关闭，桌面上就有 5 张图了。

```
camera.start_preview()
for i in range(5):
    sleep(5)
    camera.capture('/home/pi/Desktop/image%s.jpg'% i)
camera.stop_preview()
```

3.3.8　按钮控制摄像头

当按下按钮时触发 PiCamera 拍照，代码如下。

```
from gpiozero import Button
from picamera import PiCamera
from datetime import datetime
from signal import pause
button = Button(2)
camera = PiCamera()
def capture():
    datetime = datetime.now().isoformat()
    camera.capture('/home/pi/%s.jpg' % datetime)        # 保存图片
button.when_pressed = capture
pause()
```

3.3.9　拍视频

修改代码：用 start_recording( ) 和 stop_recording( ) 代替 capture( )。

```
camera.start_preview()
camera.start_recording('/home/pi/video.h264')
sleep(10)
camera.stop_recording()
camera.stop_preview()
```

运行程序，将拍摄 10s 的视频，并保存在桌面，然后关闭预览。

3.3.10　实现按钮关机

Button 类还提供了在按钮按住一段给定时间后运行函数的功能。

下面的示例是当按钮按住 2s 时，将关闭树莓派。

```
from gpiozero import Button
from subprocess import check_call
from signal import pause
def shutdown():
    check_call(['sudo','poweroff'])          # 运行 shell
shutdown_btn = Button(17, hold_time=2)      # 定义按钮以及持续时间
shutdown_btn.when_held = shutdown
pause()
```

3.3.11　PIR 人体红外感应模块

使用 PIR 传感器可以检测红外线，用来判断是否有人接近它。PIR 传感器有 3 根针脚：VCC 接 5V，OUT 接 GP4，GND 接树莓派 GND。

检测人体是否接近的程序如下：

```
from gpiozero import MotionSensor
pir = MotionSensor(4)
while 1 :
    if pir.motion_detected:
        print("Somebody")
    else:
        print("Nobody")
```

运行程序，当用手在 PIR 传感器前面挥动时，可以看到命令窗口会输出 Somebody；没有检测到人时，命令窗口会输出 Nobody。

3.4　OpenCV 编程

3.4.1　OpenCV 简介

OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

OpenCV 用 C++ 语言编写，它的主要接口也是用 C++ 语言编写，但是依然保留了大量的 C 语言接口。该库也有大量的 Python、Java、MATLAB/OCTAVE（版本 2.5）的接口。这些语言的 API 接口函数可以通过在线文档获得，如今也提供对于 C#、Ch、Ruby 的支持。

OpenCV 对非商业应用和商业应用都是免费（FREE）的。

OpenCV 提供的视觉处理算法非常丰富，致力于真实世界的实时应用，通过优化的 C 代码的编写对其执行速度带来了可观的提升。

OpenCV 的应用领域主要有：

（1）人机互动。

（2）物体识别。

（3）图像分割。

（4）人脸识别。

（5）动作识别。

（6）运动跟踪。

（7）机器人。

（8）运动分析。

（9）机器视觉。

（10）结构分析。

（11）汽车安全驾驶。

3.4.2　OpenCV 图像处理

图像处理 (Image Processing) 指用计算机对图像进行分析，以达到所需结果的技术，又称影像处理。图像处理一般指数字图像处理。

数字图像处理的一般过程如下。

（1）导入图像。

（2）去噪处理。

（3）图像增强。

（4）彩色图像转变成灰度图。

（5）灰度图转化成二值图。

（6）边缘检测 / 分割。

（7）直方图匹配 / 轮廓匹配。

OpenCV 提供的图像处理算法非常丰富。例如:图像的基本操作,包括读取、显示、存储;图像的腐蚀与膨胀；图像的色彩变换；图像的特征提取等都可以通过调用 OpenCV 中的算法函数实现。本章将介绍常用的 OpenCV 函数。

3.4.3　安装 OpenCV

登录树莓派系统，打开终端，执行如下命令：



```
$ sudo apt-get install python-opencv python-numpy
```

进入 python 命令行，查看 OpenCV 版本：

```
$ python
>>> import cv2
>>> python cv2.__version__
>>> 2.4.9.1
```

执行结果如图 3-17 所示。

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170124] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> print cv2.__version__
2.4.9.1
>>>
```

图3-17　查看OpenCV版本

3.4.4　OpenCV 库函数简介

OpenCV 包含非常丰富的函数库。这些函数被封装在不同的模块里。下面列举了部分常见的模块。

（1）core：核心函数库。

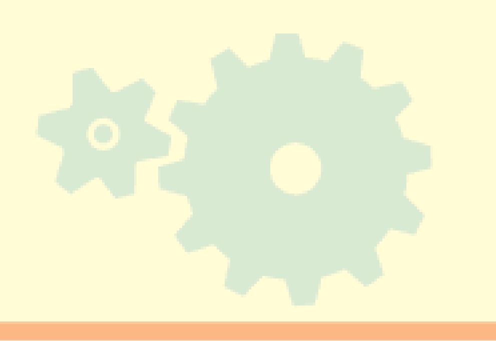
（2）imgproc：图像处理库。包含线性、非线性图像滤波器、几何图像变换（大小拉伸、仿射变换、倾斜、重映射）、颜色空间变换、直方图等。

（3）highgui：GUI、媒体 I/O 函数库。

（4）video：视频处理库。包含运动估计、去背景、对象捕捉算法等。

（5）calib3d：处理多个视角的几何算法库，包含单个、双目相机校正、对象姿势估计等。

（6）features2d：特征检测函数库。



（7）objdetect：对象检测函数库，以及一些预定义类型的实例（如脸部、眼睛、马克杯、人、车等）。

（8）ml：机器学习函数库。

（9）flann：高维空间中的聚类和搜索函数库。

（10）gpu：GPU 加速算法。

（11）stitching：图像拼接函数库。

3.4.5　读取图像

OpenCV 读取图像使用函数：imread( )；显示图像使用函数：imshow( )。

新建一个名为 show_ pic.py 的文件，输入如下程序：

```
import numpy as np
import cv2 as cv
img = cv.imread('pic/lena.png')
cv.imshow('image',img)
k = cv.waitKey(0)
if k == 27:
    cv.destroyAllWindows()
```

执行程序，结果如图 3-18 所示。

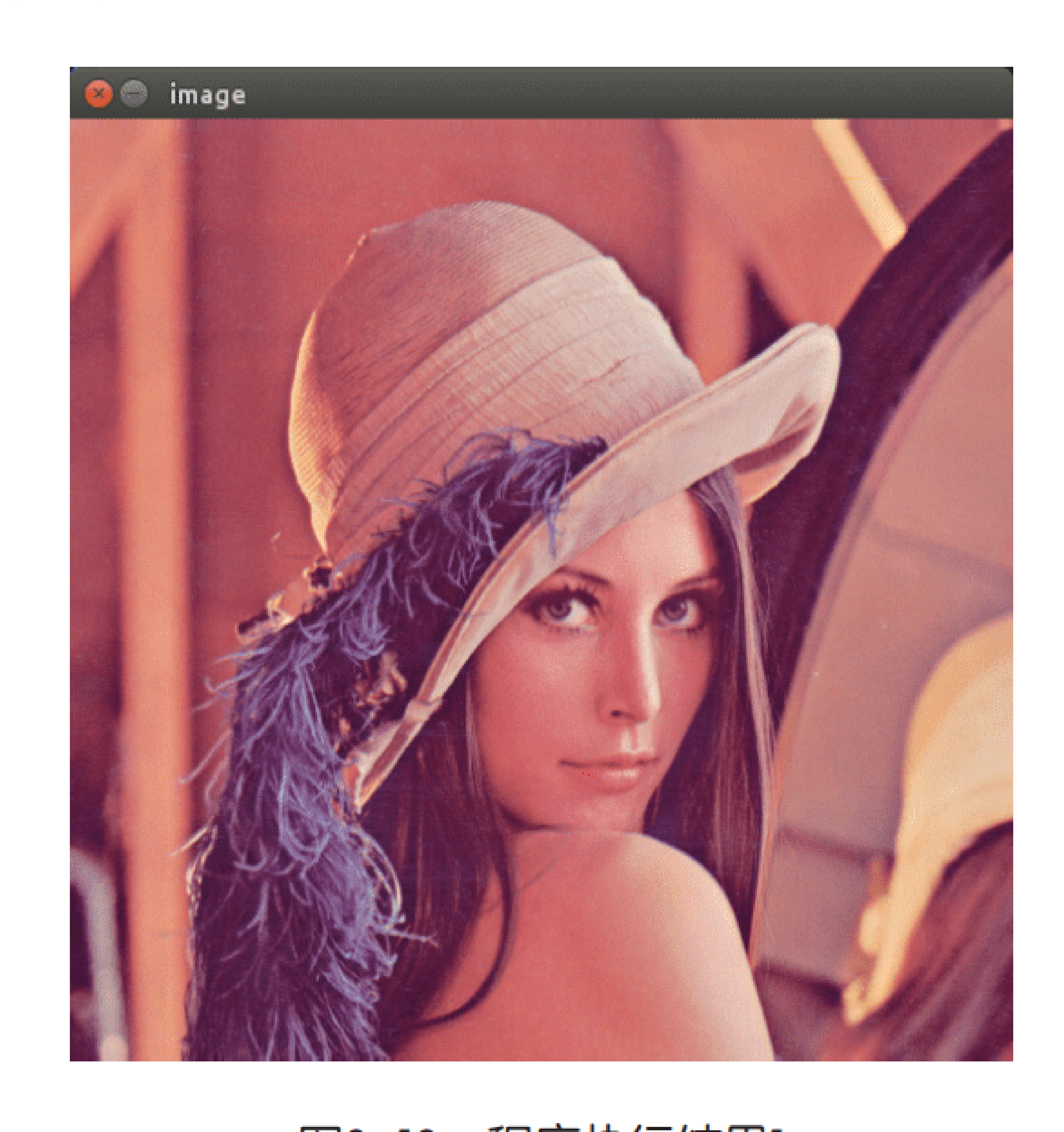


图3–18　程序执行结果1

3.4.6 将图像转换为灰度图像

在图像处理中，经常使用到灰度图像，对灰度图像进行处理，可以降低计算量并得到与处理彩色图像同样的效果。

图像转换使用到函数：cvtColor( )。

新建一个名叫 show_ pic_ gray.py 的文件，输入如下程序：

```
import numpy as np
import cv2 as cv
img = cv.imread('pic/lena.png')
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
cv.imshow('image',gray)
k = cv.waitKey(0)
if k == 27:
     cv.destroyAllWindows()
```

执行程序，结果如图 3-19 所示。

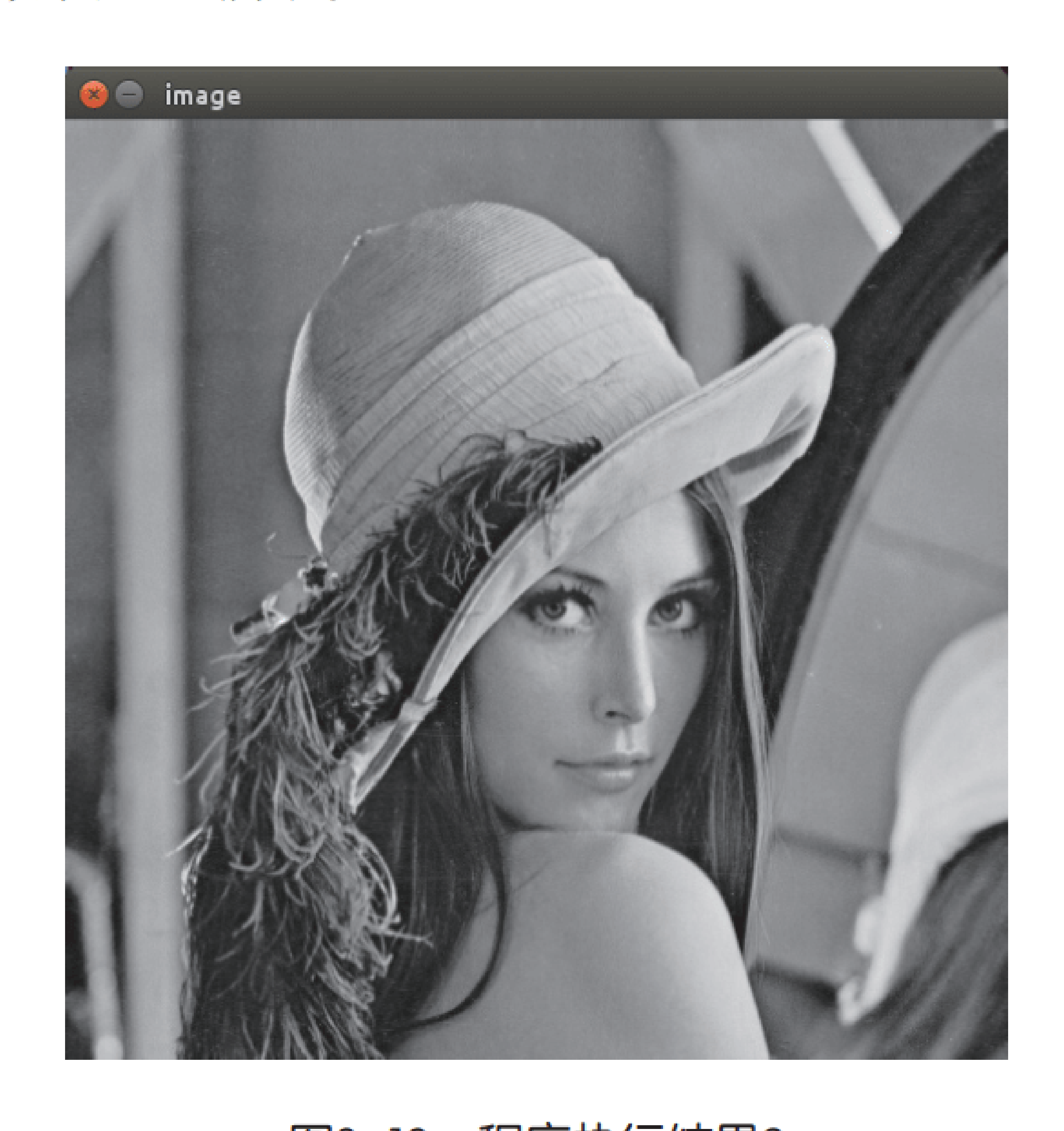


图3–19 程序执行结果2

3.4.7 画图

OpenCV 支持在图像上画图，这样可以对图像进行标记等。

line( )：画线。

circle( )：画圆。

rectangle( )：画矩形。

ellipse( )：画椭圆。

putText( )：绘制文字。

新建一个名叫 show_ pic_draw.py 的文件，输入如下程序：

```
import numpy as np
import cv2 as cv
img = cv.imread('pic/lena.png')
print img.shape
cv.line(img,(0,0),(50,50),(0,0,255),5)
cv.rectangle(img,(50,50),(100,100),(255,0,0),5)
cv.circle(img,(150,150),50,(0,255,0),5)
cv.imshow('image',img)
k = cv.waitKey(0)
if k == 27:
     cv.destroyAllWindows()
```

执行程序，结果如图 3-20 所示。

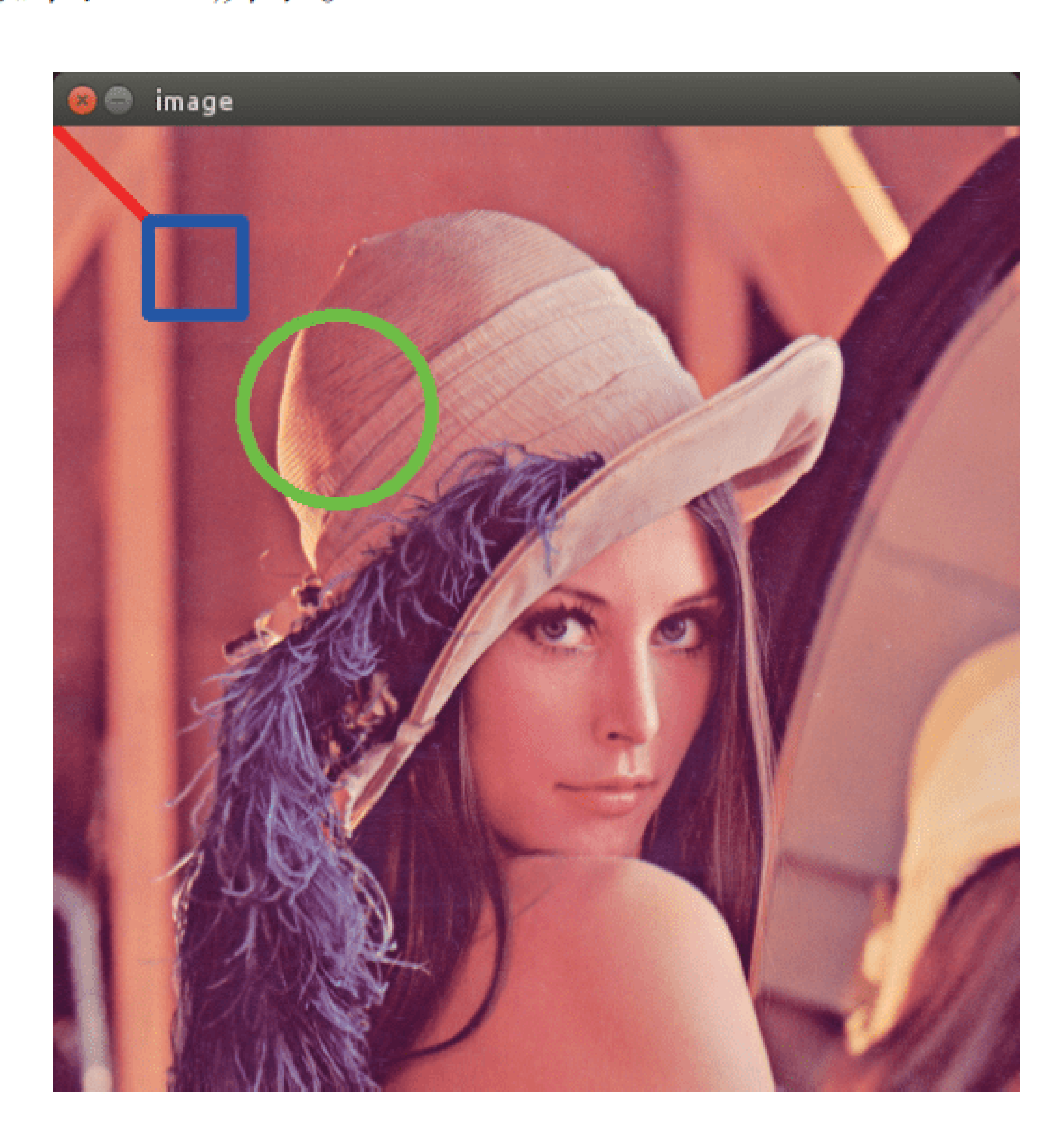


图3-20　程序执行结果3

可以看到，画了一条线、一个矩形和一个圆。

3.4.8 人脸识别

OpenCV 使用基于 Haar 特征的级联分类器，这种方法使用机器学习对大量图片进行训练，从而对物体进行识别。

新建一个名叫 face_detect.py 的文件，输入如下程序：

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
img = cv2.imread('lena.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray,1.1,5)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
cv2.imshow('img',img)
cv2.imwrite('../result.jpg',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

执行程序，结果如图 3-21 所示。

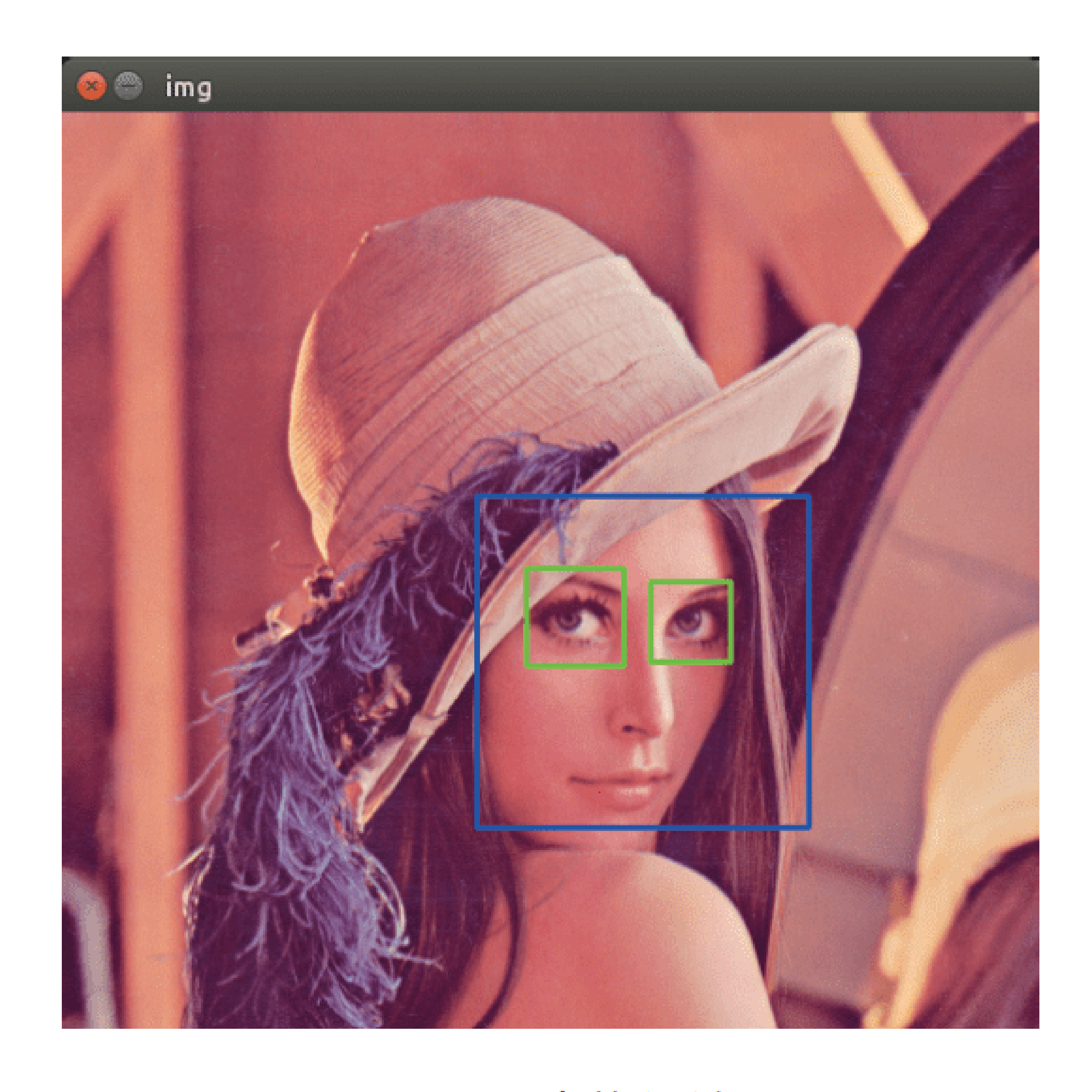


图3-21 程序执行结果4

你也可以加入诸如“眼睛检测”甚至“微笑检测”这样的检测器。在这样的应用中，你需要把分类器函数和矩形框加入原有的面部识别区域中，因为在区域外进行识别没有意义。

注：在树莓派上，分类方法（HaarCascades）会消耗大量 CPU 计算，所以在同一代码中使用多个分类器将会显著减慢处理速度。在计算机上运行这些算法则非常容易。

3.4.9 简单的图像识别——数字识别

这是一个最简单的图像识别，将图 3-22 的图片加载后直接利用 Python 的一个识别引擎进行识别，将图片中的数字通过 pytesseract.image_to_string(image) 识别并将结果显示出来。



图3-22 要识别的数字

Python 代码如下：

```
#-*-coding:utf-8-*-
import sys
reload(sys)
sys.setdefaultencoding('utf-8')
import time
time1 = time.time()
from PIL import Image
import pytesseract

########### 二值化算法
def binarizing(img,threshold):
```

```
        pixdata = img.load()
        w, h = img.size
        for y in range(h):
            for x in range(w):
                if pixdata[x, y] < threshold:
                    pixdata[x, y] = 0
                else:
                    pixdata[x, y] = 255
        return img
    image = Image.open(r'E:\taqu\12.png')
    ########### 去除干扰线算法
    def depoint(img): #input: gray image
        pixdata = img.load()
        w,h = img.size
        for y in range(1,h-1):
            for x in range(1,w-1):
                count = 0
                if pixdata[x,y-1] > 245:
                    count = count + 1
                if pixdata[x,y+1] > 245:
                    count = count + 1
                if pixdata[x-1,y] > 245:
                    count = count + 1
                if pixdata[x+1,y] > 245:
                    count = count + 1
                if count > 2:
                    pixdata[x,y] = 255
        return img
    ######## 转化为灰度图
    img = image.convert('L')
    ######## 把图片变成二值图像
    img1=binarizing(img,190)
    # img2=depoint(img1)
    img1.show()
    code = pytesseract.image_to_string(img1)
    print "识别该验证码是 :" + str(code)
```

想要实现上面的代码需要安装两个包和一个引擎。

（1）pip install pytesseract：安装图像识别库。

（2）pip install pillow：Python 平台上的图像处理标准库，功能非常强大。

（3）安装识别引擎 tesseract-ocr 与汉化包，光学字符识别引擎。

安装完成后需要配置环境变量，在系统变量 path 后增加 tesseract-ocr 的安装地址 C:\Program Files (x86)\Tesseract-OCR。

一切都安装完成后运行上述代码，会发现报错，此时需要在 Python 安装文件夹下的 lib\site-packages\pytesseract 下找到 pytesseract.py，做如下修改：

```
Tesseract_cmd='C:/Program Files (x86)/Tesseract-OCR/tesseract.exe'
```

运行结果如图 3-23 所示。图片识别成功，但识别率只有约 90%。

```
"D:\Program Files\Python27\python.exe" D:/PycharmProjects/learn2017/简单验证码识别.py
识别该验证码是:7364

Process finished with exit code 0
```

图3–23　程序执行结果5

3.4.10　简单的图像识别——英文识别

识别图 3-24 中的英文。

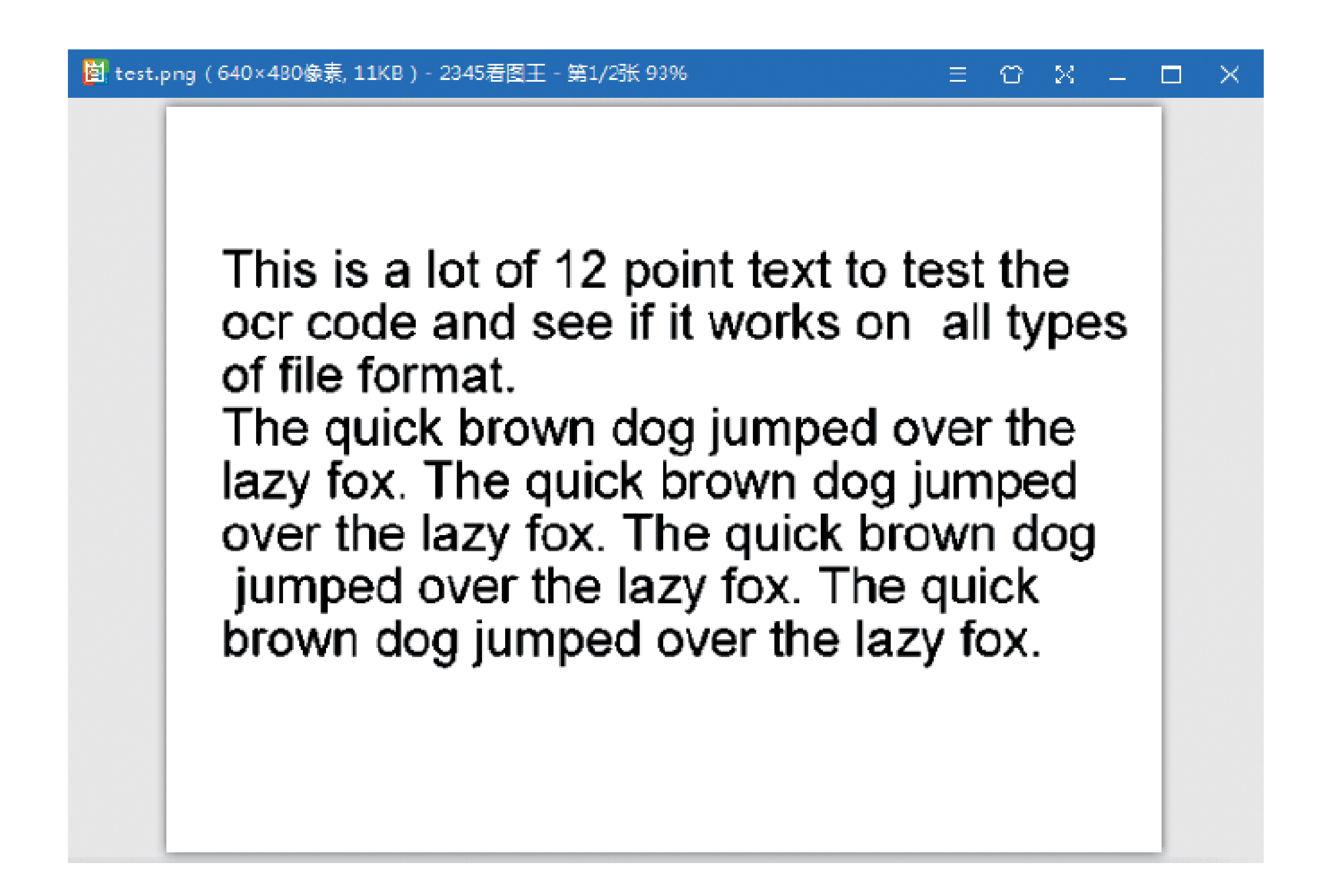


图3–24　要识别的英文

Python 程序如下：

```
#-*-coding:utf-8-*-
```




```
import sys
reload(sys)
sys.setdefaultencoding('utf-8')
import time
time1 = time.time()
from PIL import Image
import pytesseract
image = Image.open(r'D:\Program Files\Python27\Lib\site-packages\
pytesseract\test.png')
code = pytesseract.image_to_string(image)
print(code)
```

运行结果如图 3-25 所示，图片识别成功。

```
Run 简单验证码识别
"D:\Program Files\Python27\python.exe" D:/PycharmProjects/learn2017/简单验证码识别.py
This is a lot of 12 point text to test the
ocr code and see if it works on all types
of file format.

The quick brown dog jumped over the
lazy fox. The quick brown dog jumped
over the lazy fox. The quick brown dog
jumped over the lazy fox. The quick
brown dog jumped over the lazy fox.

Process finished with exit code 0
```

图3-25　程序执行结果6

3.5　天气预报小程序

通过树莓派，可以实时获取天气预报数据，所以开始编程前要确保将树莓派连接到网络中。目前树莓派获取免费天气信息并解码输出的方法有很多种，本节将介绍一种最简单的方法，实现一个天气预报小程序，实时播报北京的天气。有兴趣的读者也可以尝试其他方法。

在 http://www.weather.com.cn 中搜索你所要的城市。进入天气页之后，网址末尾的数字就是城市代码。例如：北京天气网址是 http://www.weather.com.cn/weather1d/101010100.shtml#input，这里的数字 101010100 就是北京的城市代码。

Python 程序如下：

```
import urllib.request
import json
```



```
ApiUrl="http://www.weather.com.cn/data/sk/101010100.html"
html=urllib.request.urlopen(ApiUrl)
data=html.read().decode("utf-8")   # 读取并解码
ss=json.loads(data)                # 将 JSON 编码的字符串转换回 Python 数据结构
info=ss['weatherinfo']
print('城市：%s'%info['city'])
print('温度：%s 摄氏度'%info['temp'])
print('风速：%s'%info['WD'],info['WS'])
print('湿度：%s'%info['SD'])
print('时间：%s'%info['time'])
```

运行程序，结果如下。

```
城市：北京
温度：27.9 摄氏度
风速：南风小于 3 级
湿度：28%
时间：17:55
```





第 4 章 树莓派与 MegaPi 结合

4.1 树莓派与MegaPi连接与通信

4.1.1 连接树莓派和 MegaPi

MegaPi 的 USB 通信端口旁边预留了 10 针的树莓派连接端口，可以通过 MegaPi 套装附件中的 10 针排线连接树莓派，如图 4-1 和图 4-2 所示。

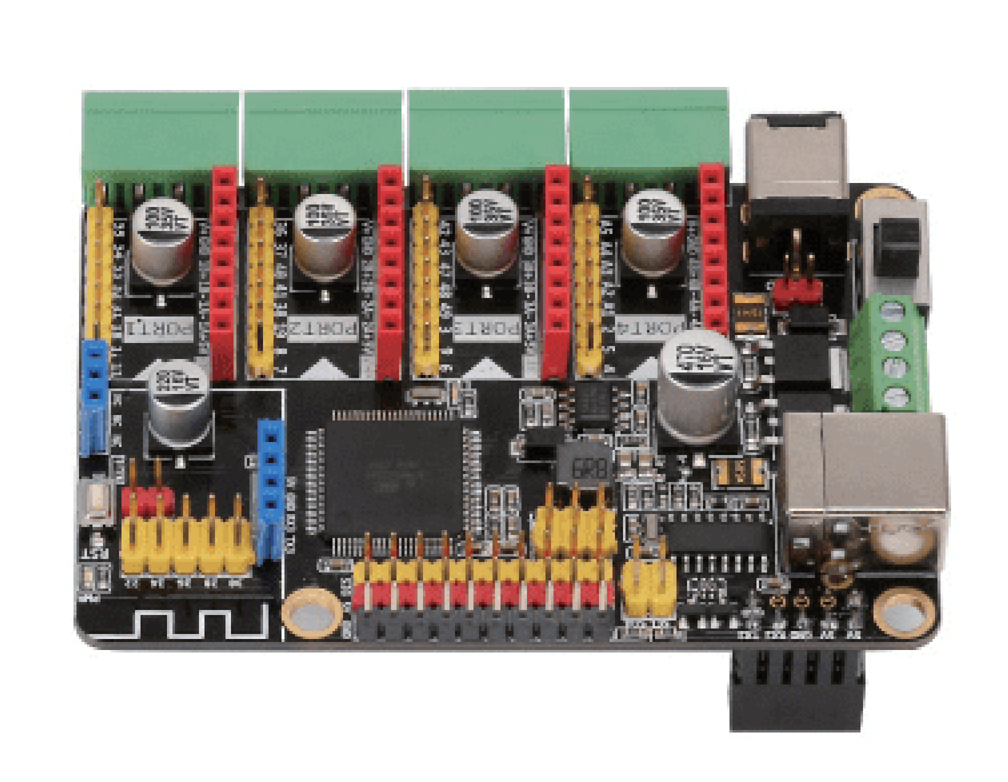


图4–1 MegaPi

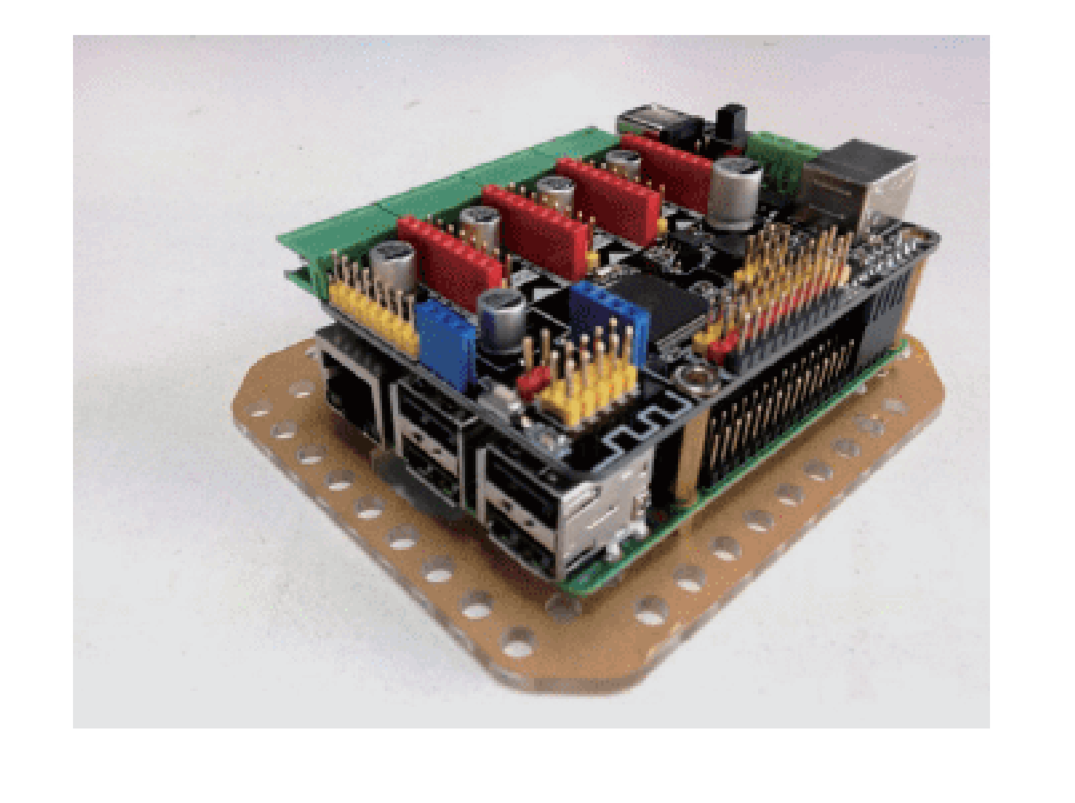


图4–2 树莓派和MegaPi连接

将树莓派和 MegaPi 通过连接端口接在一起，并用铜柱固定。

MegaPi 连接到树莓派的 GPIO 引脚如表 4-1 所示，可以看到树莓派与 MegaPi 的通信是由 GPIO 串口完成的。

表 4–1 树莓派和 MegaPi 连接引脚定义

功能名	物理引脚 BOARD编码		功能名
3.3V	1	2	5V
SDA.1	3	4	5V
SCL.1	5	6	GND
GPIO.7	7	8	TXD
GND	9	10	RXD



树莓派 CPU 内部有两个串口：一个是硬件串口 (官方称为 PL011 UART) ; 另一个是迷你串口 (官方称为 mini-UART)。官方在树莓派 3 的设计上，将硬件串口分配给新增的蓝牙模块，而将一个没有时钟源、必须由内核提供时钟参考源的“迷你串口”分配给 GPIO 的串口，这样一来由于内核的频率是变化的，会导致“迷你串口”速率不稳定，从而出现了无法正常使用的情况。

目前解决的方法是，关闭蓝牙对硬件串口的使用，将硬件串口重新恢复给 GPIO 的串口使用。这就意味着树莓派 3 的板载蓝牙和串口暂时无法使用。

4.1.2　树莓派恢复硬件串口的方法

（1）下载 pi3-miniuart-bt-overlay 文件，解压出 pi3-miniuart-bt-overlay.dtb 文件，并将 dtb 文件复制到 /boot/overlays/ 目录下。

（2）编辑 /boot 目录下的 config.txt 文件：sudonano /boot/config.txt。

添加下面两行程序，如图 4-3 所示。

```
dtoverlay=pi3-miniuart-bt-overlay
force_turbo=1
```

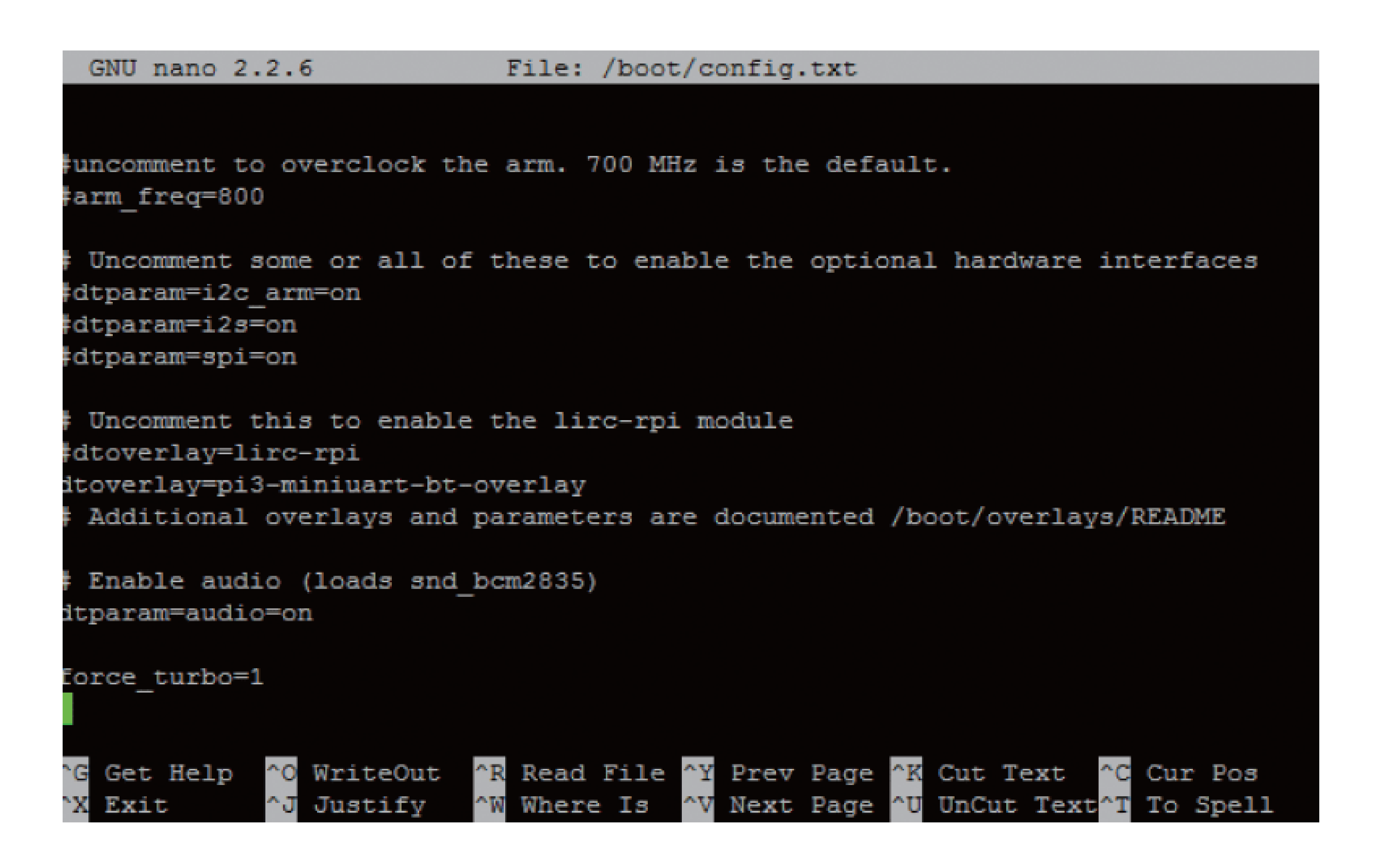


图4-3　恢复硬件串口

（3）编辑 /boot 目录下的 cmdline.txt 文件。

参考下面内容修改：



```
dwc_otg.lpm_enable=0
console=serial1,115200
console=tty1
root=/dev/mmcblk0p2
kgdboc=serial1,115200
rootfstype=ext4
elevator=deadline
fsck.repair=yes
rootwait
```

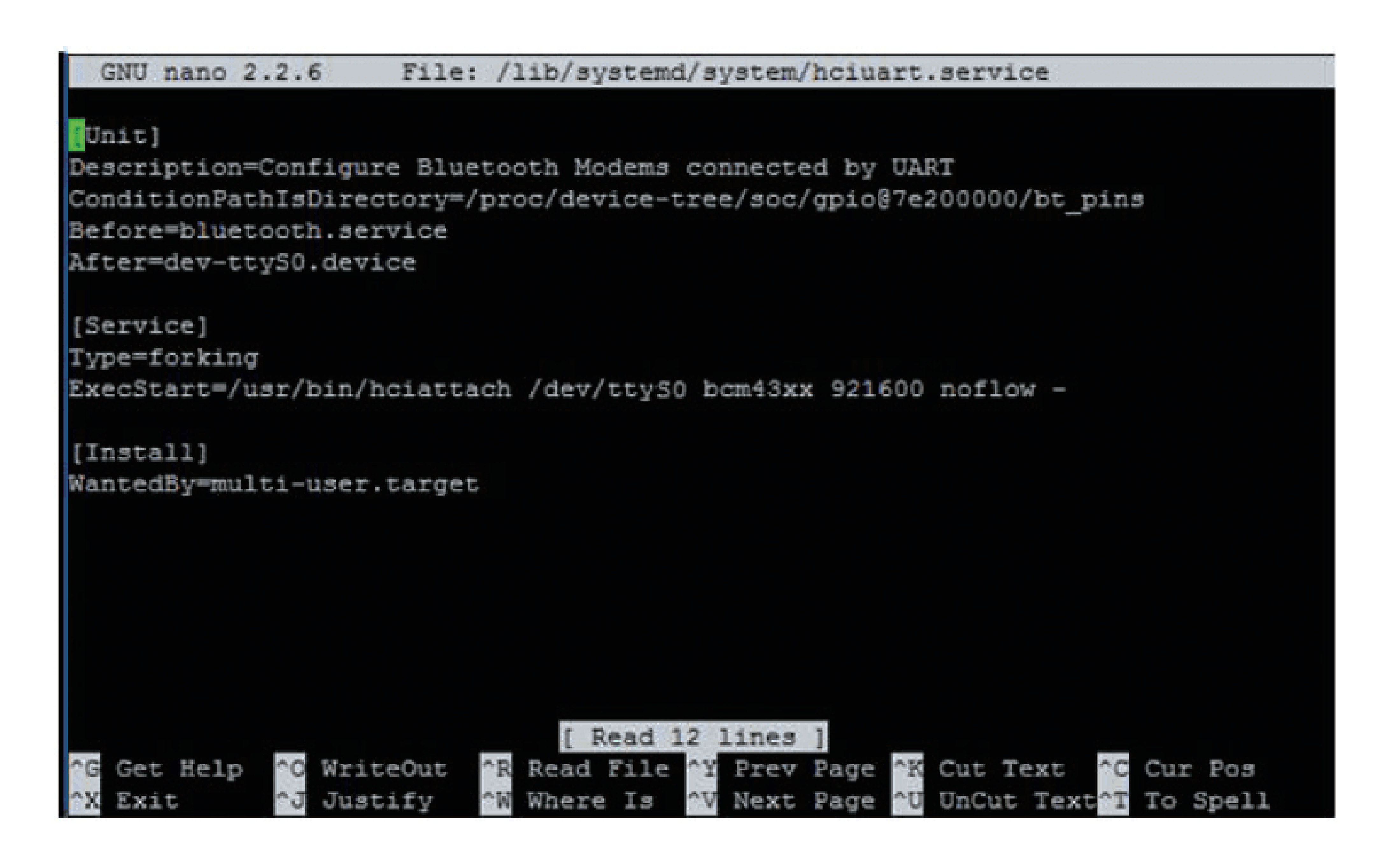
保存之后退出。

4.1.3 树莓派关闭板载蓝牙的方法

（1）SSH 登录树莓派 3 后，输入下面命令关闭 hciuart，使用 uart0：

```
sudo systemctl disable hciuart
```

（2）编辑 /lib/systemd/system 目录下的 hciuart.service 文件：sudonano/lib/systemd/system/hciuart.service，将文档中 2 处 ttyAMA0 修改为 ttyS0，如图 4-4 所示。


```
  GNU nano 2.2.6       File: /lib/systemd/system/hciuart.service

[Unit]
Description=Configure Bluetooth Modems connected by UART
ConditionPathIsDirectory=/proc/device-tree/soc/gpio@7e200000/bt_pins
Before=bluetooth.service
After=dev-ttyS0.device

[Service]
Type=forking
ExecStart=/usr/bin/hciattach /dev/ttyS0 bcm43xx 921600 noflow -

[Install]
WantedBy=multi-user.target

                              [ Read 12 lines ]
^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

图4-4 关闭板载蓝牙

保存之后退出，更新并重启。现在就可以通过 ttys0 与外部设备进行连接了。

4.1.4 树莓派串口测试

Python 编程要用到 serial 扩展库，需要使用如下命令先安装好。

```
sudo apt-get install python-serial
```

下面是一个非常简单的串口程序：树莓派通过串口返回接收的内容。

```
# -*- coding: utf-8-*
import serial
import time
# 打开串口
ser = serial.Serial("/dev/ttyS0", 9600)
def main():
    while True:
        count = ser.inWaiting()          # 获得接收缓冲区字符
        if count != 0:
            recv = ser.read(count)       # 读取内容并回显
            ser.write(recv)
            ser.flushInput()             # 清空接收缓冲区
            time.sleep(0.1)              # 必要的软件延时
    if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        if ser != None:
            ser.close()
```


4.1.5　MegaPi 准备

在下面的地址下载 Makeblock 的 Arduino 库文件：

https://github.com/Makeblock-official/Makeblock-Libraries/archive/master.zip

解压下载的文件，并复制 Makeblock 文件夹到 Arduino 默认库文件夹。

对于 Windows 用户，该文件夹放在 X:\Users\XXX\Documents\Arduino\libraries\。

对于 Linux 用户，该文件夹放在 /home/user_name/Arduino/libraries/。

打开 Arduino IDE，选择菜单命令 File → Examples，从中选取 megapi_firmware 固件，如图 4-5 所示。

该固件位置位于 Examples → MakeBlockDrive → Firmware_For_mBlock。

将该固件烧写到 MegaPi 上，至此，MegaPi 的工作已经做完了。



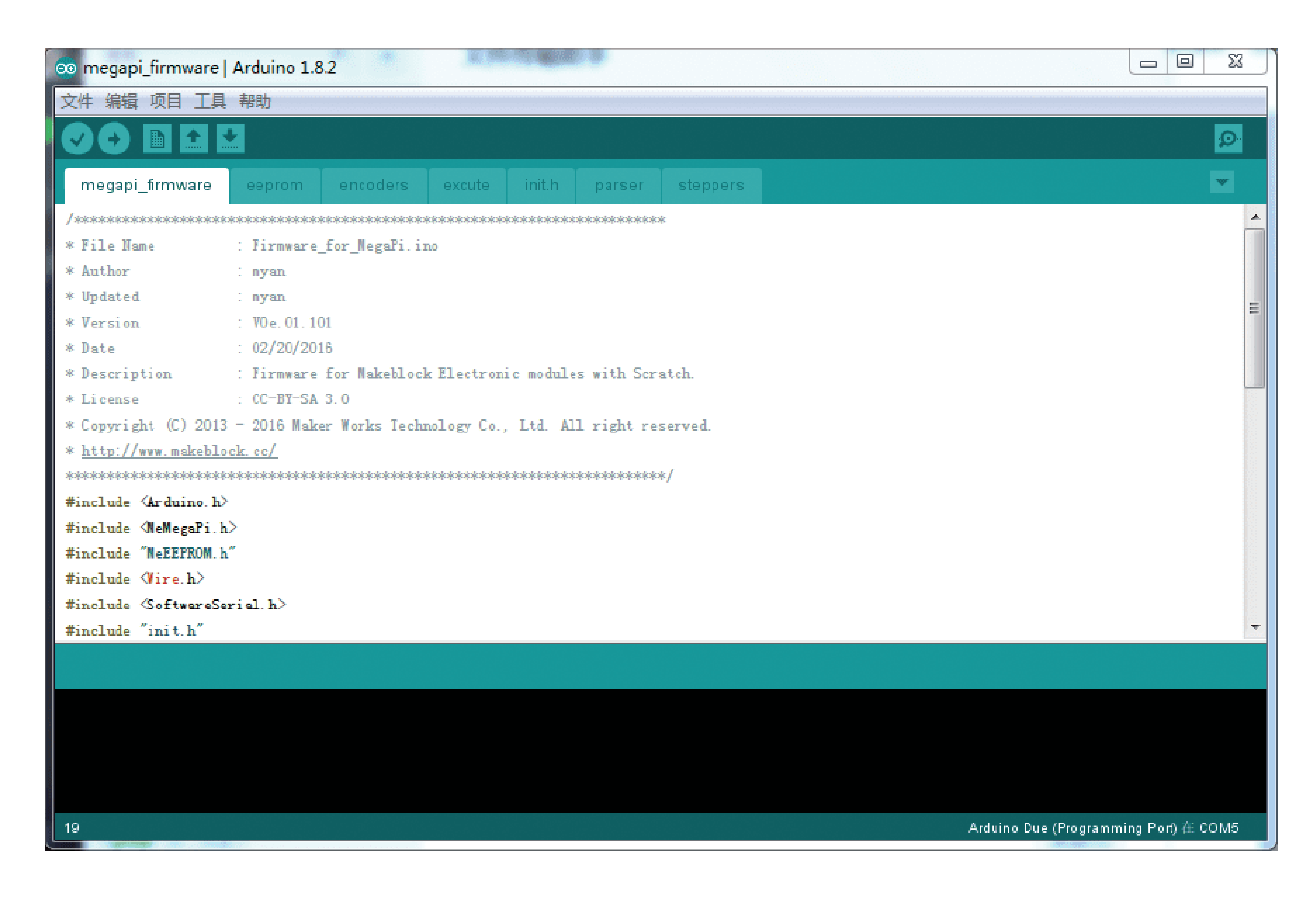


图4–5 下载固件

4.2 树莓派与MegaPi结合实例

4.2.1 控制舵机

（1）连接舵机，如图 4-6 所示。

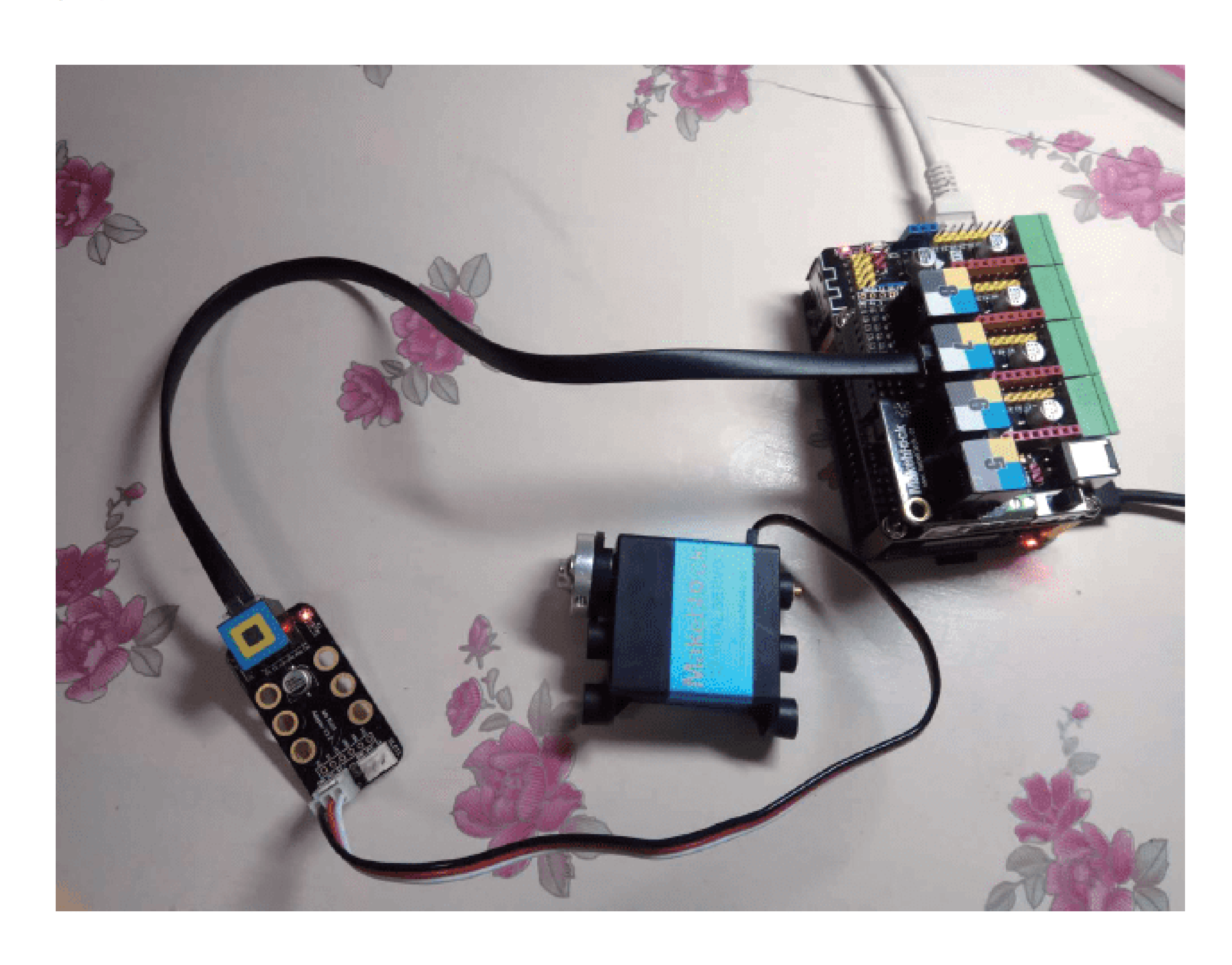


图4–6 连接舵机示意

舵机接 RJ45 转接板第二个插槽，RJ45 转接板接 MegaPi 的 Port7。

Python 程序如下。

（1）打开树莓派的终端，新建一个文件夹，命名为 MegaPi：

```
$ mkdir MegaPi
```

（2）新建一个文件，命名为 servo_test.py：

```
$ touch servo_test.py
```

（3）向该文件写入如下代码：

```
from MegaPi import *
if __name__ =='__main__'
bot = MegaPi()
bot.start('/dev/ttyS0')
while True:
     sleep(1)
     bot.servoRun(7,2,0)
     sleep(1)
     bot.servoRun(7,2,180)
```

程序说明如下。

（1）MegaPi 是一个 Python 库，实现了一个类：MegaPi，这个类的实例也就是上述代码中的：

```
Bot=MegaPi()
```

这个实例相当于一个管理器，用来管理树莓派和 MegaPi 板子的所有事件，包括树莓派连接 MegaPi、读取 MegaPi 采集到的数据、向 MegaPi 写入数据，写入的数据通常为一些控制指令。

（2）Bot.start('/dev/ttyS0')。Bot 调用 MegaPi 类的 start 方法，建立一个和 MegaPi 板子通信的连接。这个连接函数的参数为 port，意思是通过这个端口和板子进行连接，树莓派 3 和 MegaPi 通信是通过串口 0 进行通信的，这是由硬件决定的，因此它的参数为 /dev/ttyS0。

（3）简单介绍一下 /dev/ttyS0。我们可以通过在树莓派的终端中执行如下命令：

```
$ ls /dev/ttyS*
```

查看树莓派上面的串口设备。

（4）程序接下来执行一个循环，Bot 调用一个叫 servoRun 的函数，该函数的定义在如下链接中：



```
https://github.com/Makeblock-official/PythonForMegaPi/blob/
master/src/megapi.py#L207
Def servoRun(self,port,slot,angle)
    Self.__writePackage(bytearray([0xff,0x55,0x6,0x2,0xb,port,slo
    t,angle]))
```

该函数的主要参数有 port、slot、angle。

port: MegaPi 板子上嵌套的转接板的端口号，上面标记着 5、6、7、8 的 RJ45 端口，这个端口号用来表明我们使用哪个端口，例如 port=7，表示应该将连接舵机的 RJ45 头插入转接板标记为 7 的端口。

slot: 连接舵机的转接板上有 2 个插槽，上面标记了 1 和 2，这个转接板可以连接 2 个舵机，所以我们可以控制 2 个舵机，例如 slot=1，那么就把舵机接到 1 的插槽里。

angle: 表示舵机转的角度，范围为 0° ~180° 。

4.2.2　控制 DC Motor

DC Motor 在 Makeblock 平台中是一种应用非常广泛的电动机。通过 Makeblock 中的 DC Motor-25 支架，该电动机可以很容易连接到主板。电路连接方式如图 4-7 所示。

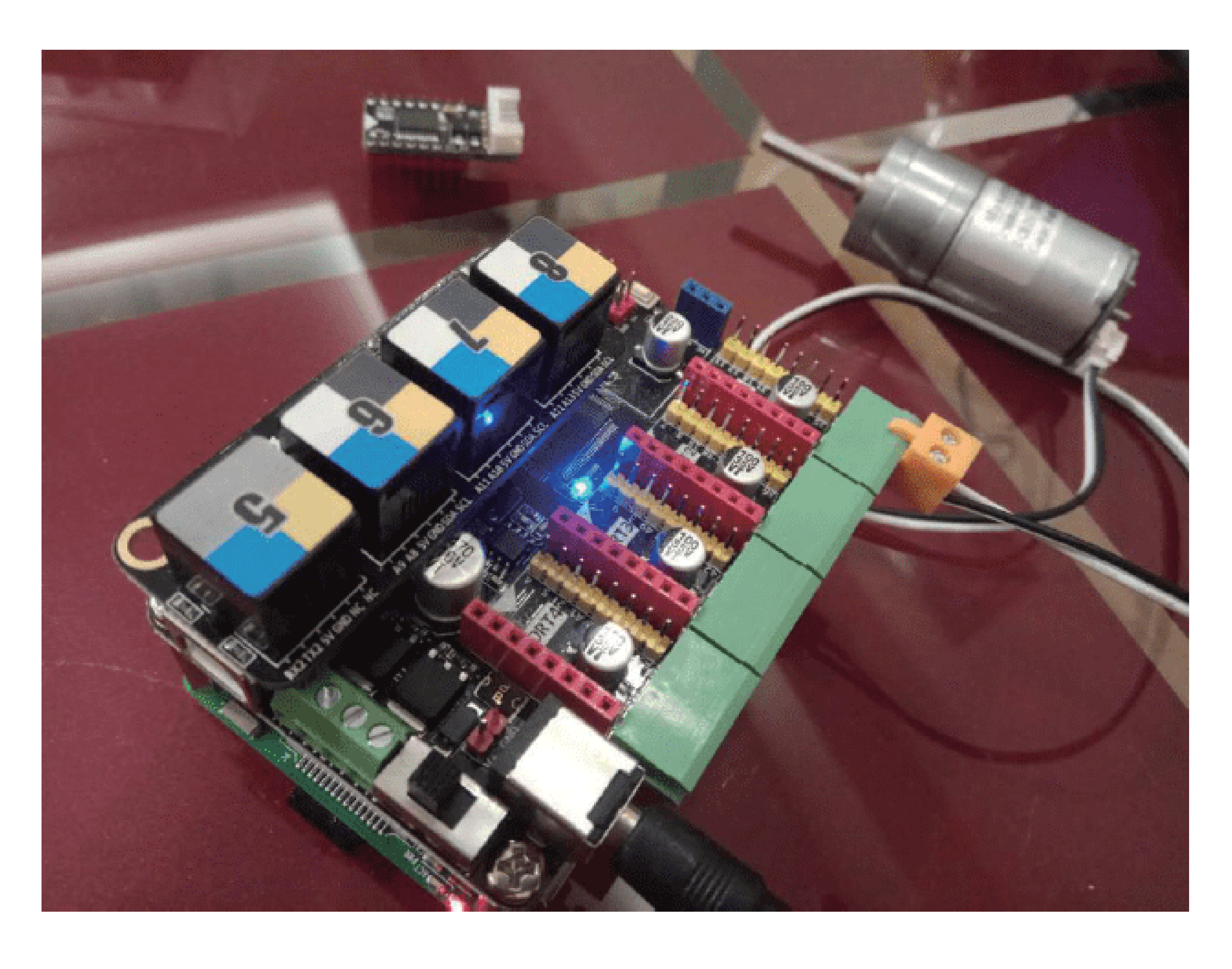


图4–7　树莓派+MegaPi控制电动机

将 DC Motor 的电动机驱动插入端口 1，并将电动机连接端插入端口 1 绿色输出端的右侧输出口。

登录树莓派，如图 4-8 所示。

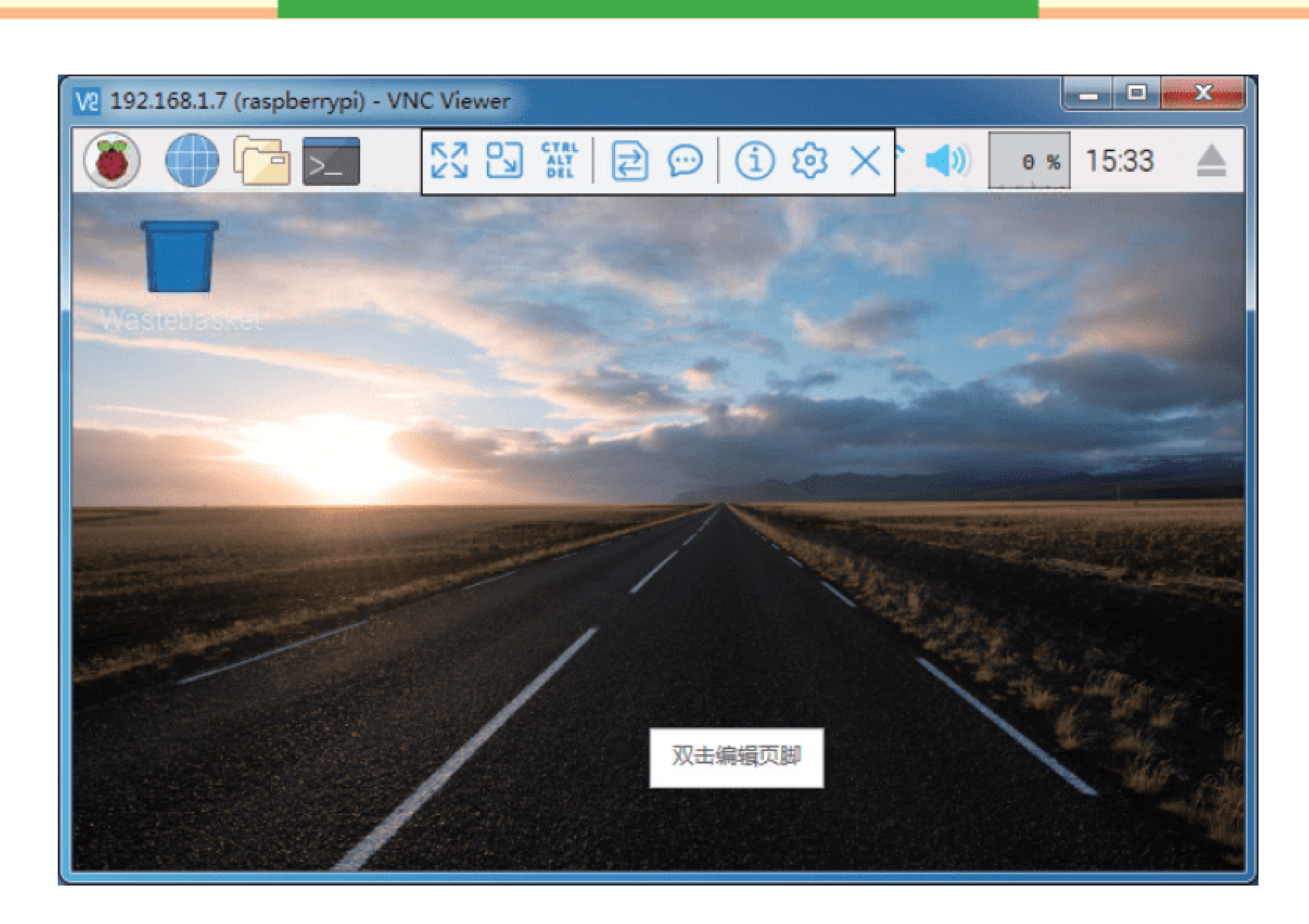


图4-8　登录树莓派

打开控制终端，如图 4-9 所示。

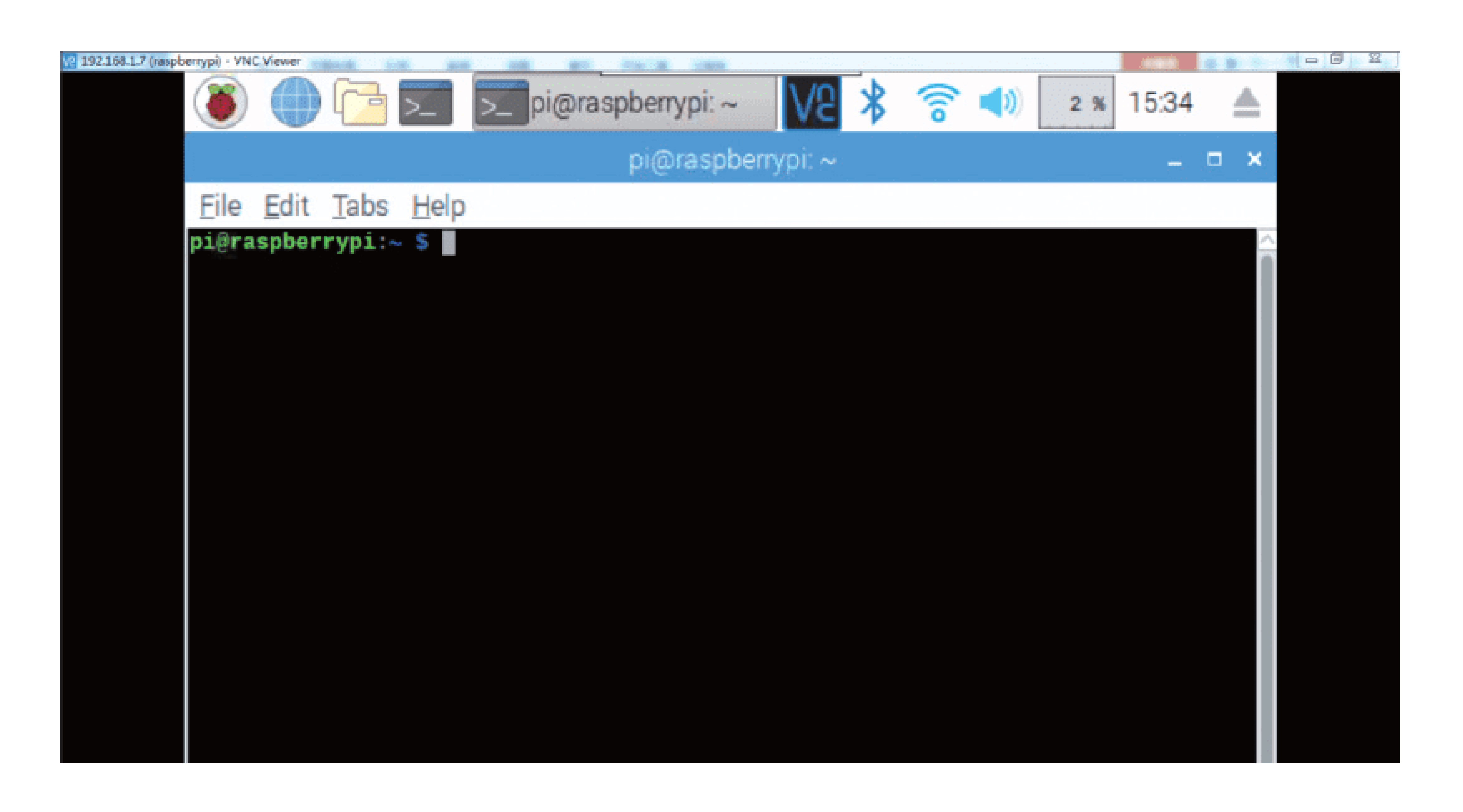


图4-9　控制终端

Python 代码如下。

```
from megapi import *
if __name__ =='__main__':
    bot = MegaPi()
    bot = start("/dev/ttyS0")
sleep(1)
while(1):
    sleep(1)
```

```
        bot.motorRun(1,50)
        sleep(1)
        bot.motorRun(1,-50)
```

程序执行结果如图 4-10 所示。

```
pi@raspberrypi: ~/maker_source_codes/PythonForMegaPi/examples
pi@raspberrypi:~/maker_source_codes/PythonForMegaPi/examples $ cat motorRun.py
from megapi import *

if __name__ == '__main__':
        bot = MegaPi()
        bot.start("/dev/ttyS0")
        sleep(1);
        while 1:
                sleep(1);
                bot.motorRun(1,50);
                sleep(1);
                bot.motorRun(1,-50);
pi@raspberrypi:~/maker_source_codes/PythonForMegaPi/examples $
```

图4–10　程序执行结果1

程序说明如下。

```
def motorRun(self, port, speed)
```

该函数是一个可以驱动 DC 电动机的函数，port 表示连接端口，speed 表示速率。

执行程序（见图 4-11），可以看到电动机每隔 1s 做正反转交替。

```
pi@raspberrypi: ~/maker_source_codes/PythonForMegaPi/examples
pi@raspberrypi:~/maker_source_codes/PythonForMegaPi/examples $ python motorRun.py
```

图4–11　电动机控制程序执行

4.2.3　烟雾传感器

常见的烟雾传感器型号为 Me Gas Sensor，它包含一个 MQ2 烟雾传感器，该传感器具有可重复性，稳定性长久，反应灵敏，性能持久。常用在家庭和工厂中的烟雾检测设备中。

连接方式如图 4-12 所示。

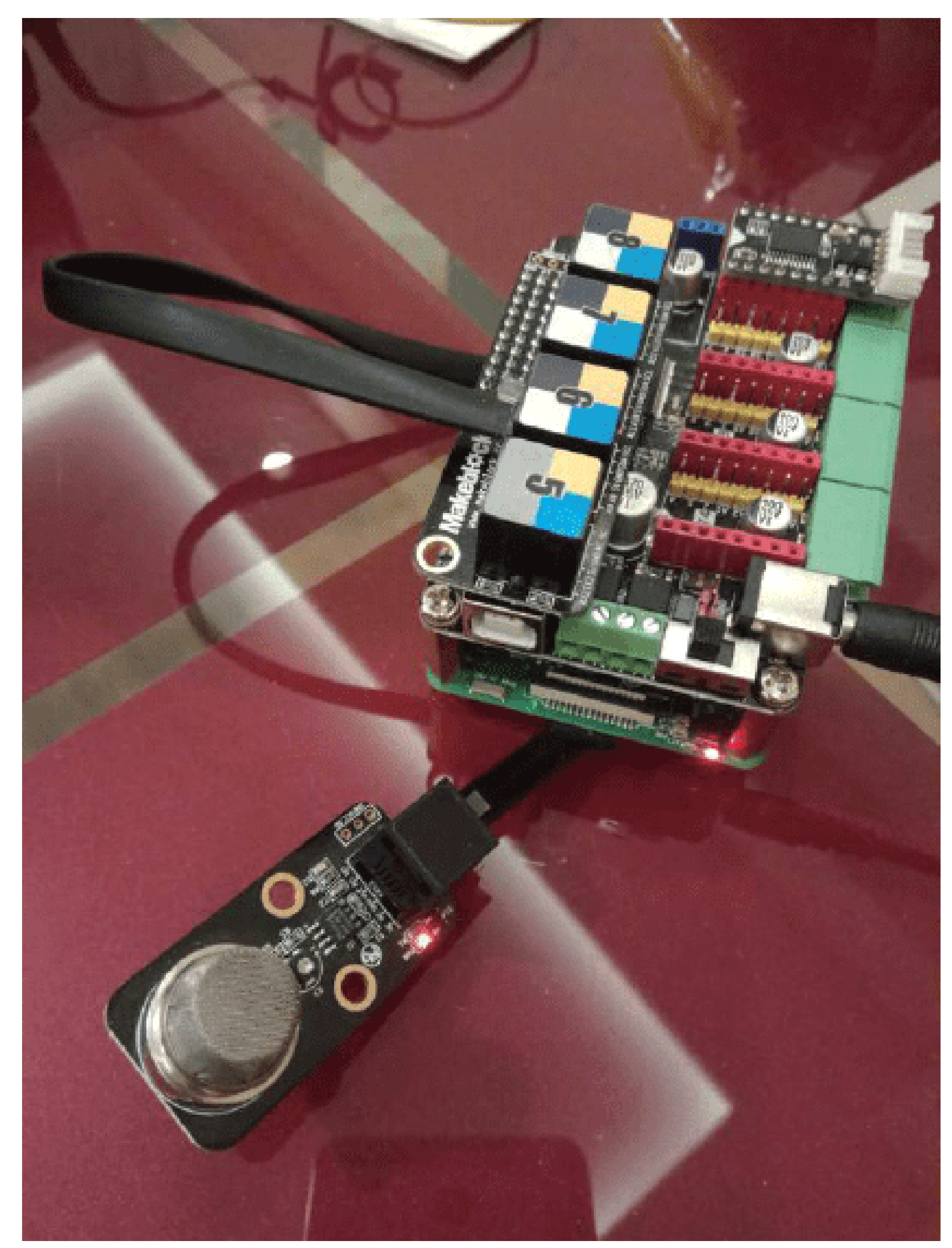


图4–12　烟雾传感器与MegaPi电路连接

实时检测烟雾浓度的 Python 代码如下。

```
from megapi import *
import signal
def sigint_handler(signum, frame):
    global is_sigint_up
    is_sigint_up = True
def onRead(a):
    print "gas:" + str(a)

signal.signal(signal.SIGTERM, sigint_handler)
signal.signal(signal.SIGINT, sigint_handler)
signal.signal(signal.SIGHUP, sigint_handler)
is_sigint_up = False
if __name__ =='__main__'
    bot = MegaPi()
    bot.start('/dev/ttyS0')
    while 1:
        bot.gasSensorRead(6,onRead)
        if is_sigint_up:
            break
```

程序执行结果如图 4-13 所示。



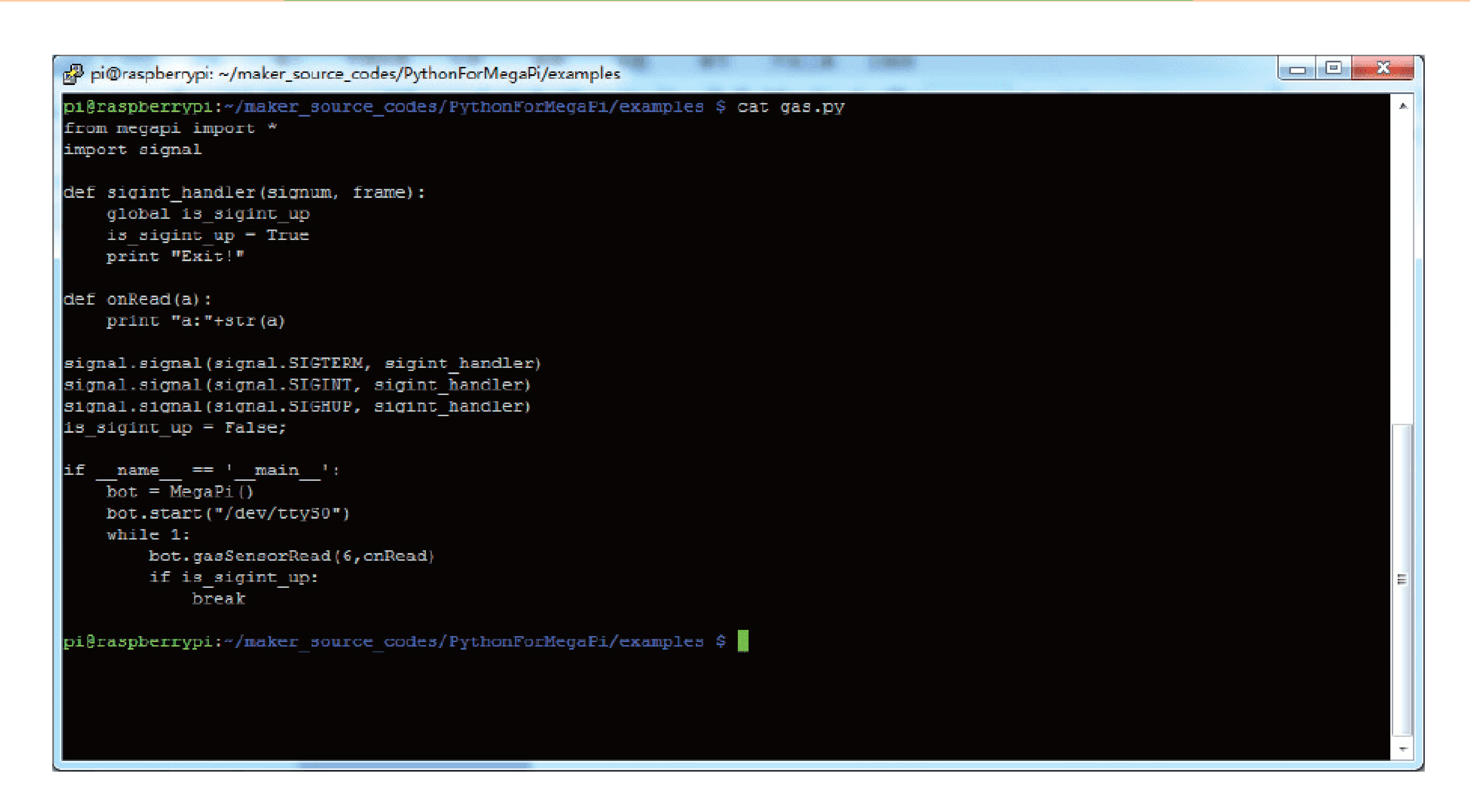


图4-13　程序执行结果2

程序说明如下。

（1）核心函数为

```
def gasSensorRead(self, port, callback)
```

其中，port 表示端口号；callback: 表示回调函数。

onRead( ) 函数传给 gasSensorRead 后，当 MegaPi 板子检测到气体数据，传给树莓派，自动调用回调函数 onRead，并且把采集到的气体数据打印出来，输出数值随检测环境中气体浓度的升高而增大，如图 4-14 所示。

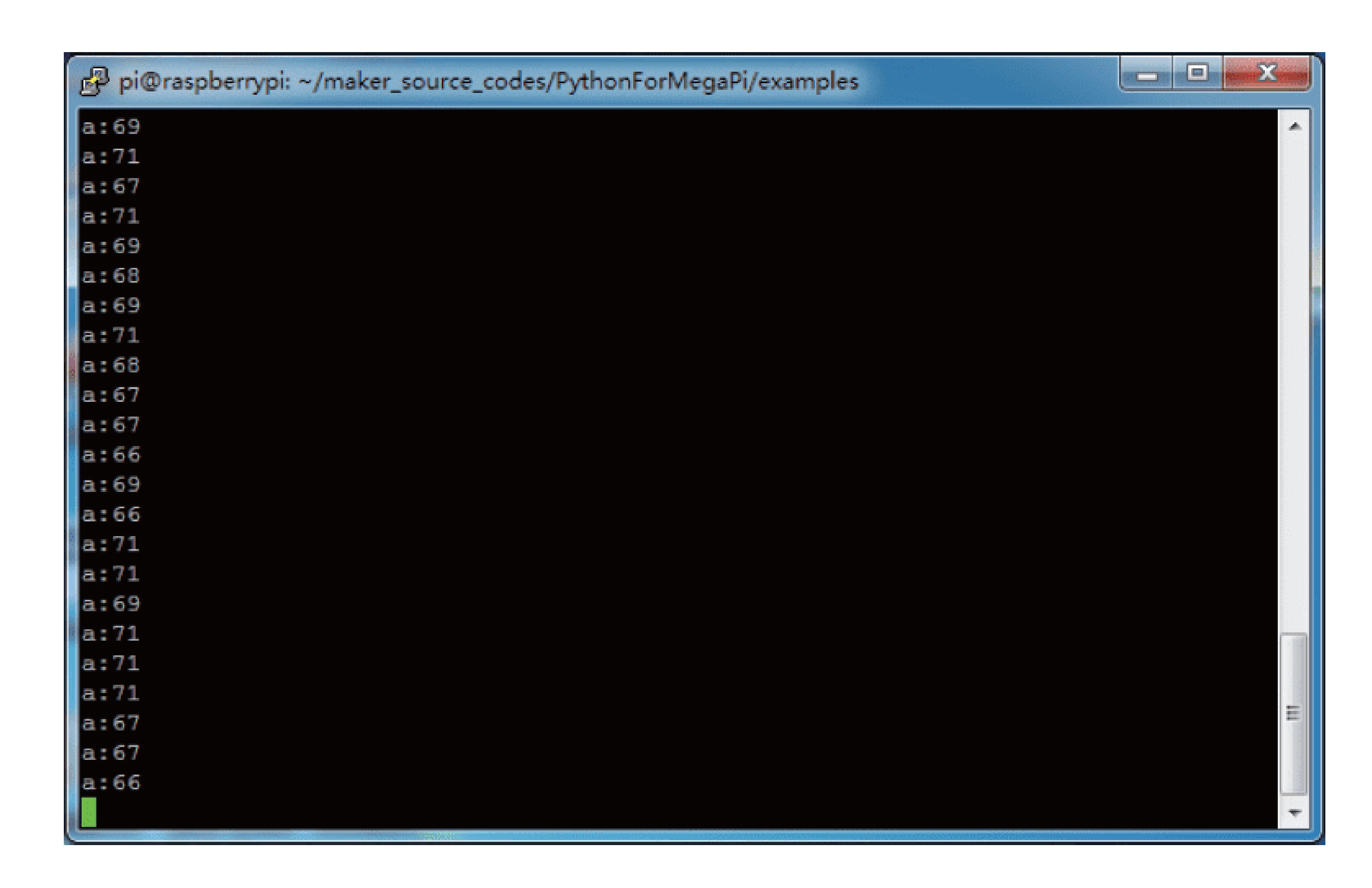
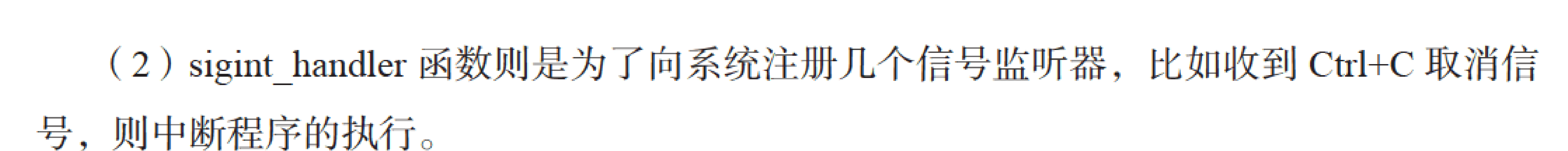


图4-14　程序执行结果3



（2）sigint_handler 函数则是为了向系统注册几个信号监听器，比如收到 Ctrl+C 取消信号，则中断程序的执行。

4.2.4　超声波传感器

超声波传感器是一种用来测量距离的传感器模块，它的有效范围为 3~400cm。该模块可以进行距离测量，用来帮助小车躲避障碍物。

传感器规格如下。

（1）工作电压：5V。

（2）测量角度：30°。

（3）测量距离：3~ 400cm。

（4）频率：42kHz。

连接方式如图 4-15 所示。

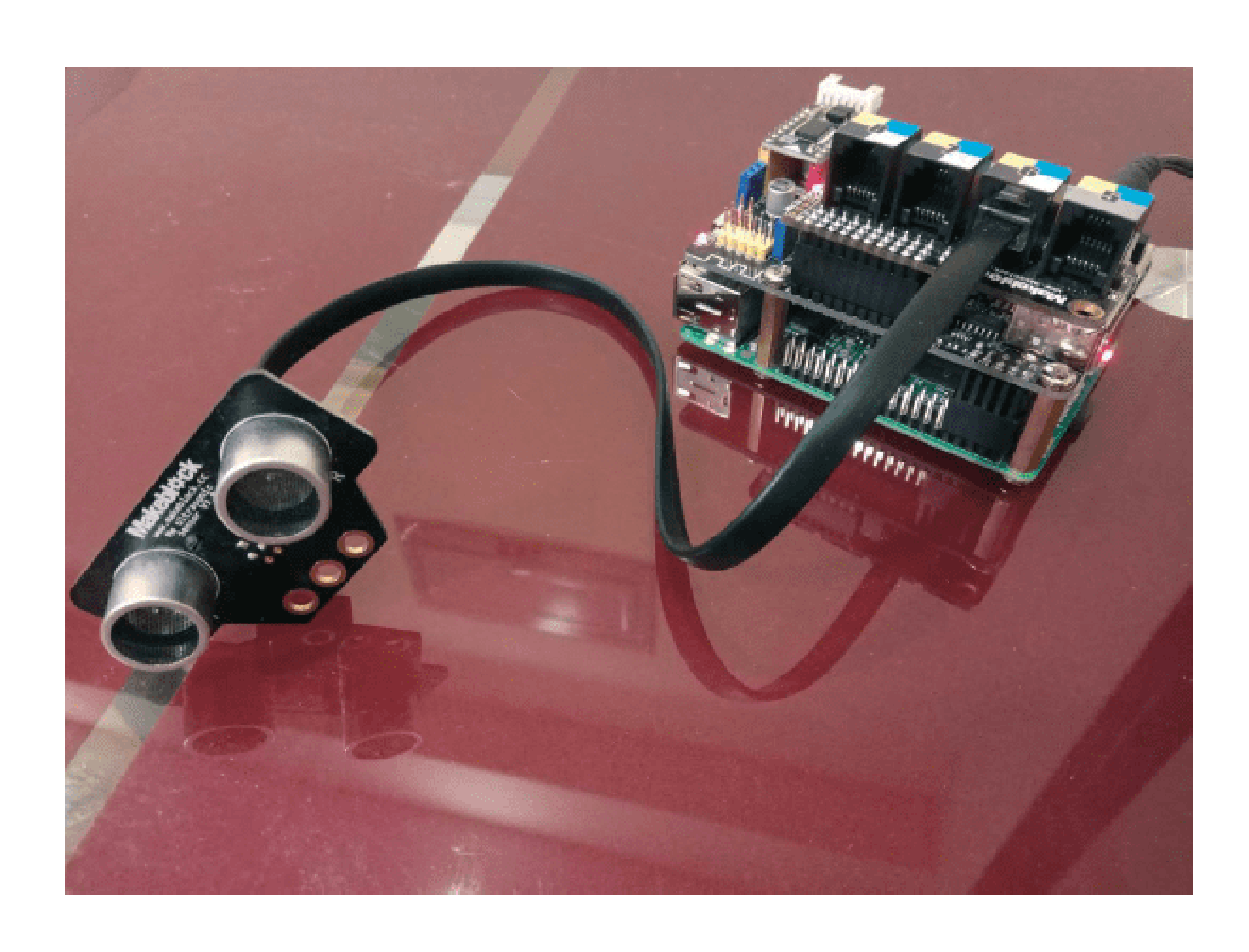


图4-15　超声波传感器与MegaPi的电路连接

超声波传感器测距的 Python 代码如下。

```
from megapi import *
import signal
def onRead(v):
    print("distance:", str(v),"cm")
def sigint_handler(signum, frame):
    global is_sigint_up
    is_sigint_up = True
    print "Exit!"
```



```
signal.signal(signal.SIGTERM, sigint_handler)
signal.signal(signal.SIGINT, sigint_handler)
signal.signal(signal.SIGHUP, sigint_handler)
is_sigint_up = False
if __name__ =='__main__':
    bot = MegaPi()
    bot.start('/dev/ttyS0')
    while 1:
        bot.ultrasonicSensorRead(6,onRead)
        if is_sigint_up:
            break
```

运行程序，结果如图 4-16 所示。

```
pi@raspberrypi: ~/maker_source_codes/PythonForMegaPi/examples
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7241363525 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.6379318237 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7586212158 cm
distance:74.7413787842 cm
distance:74.6379318237 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
distance:74.6379318237 cm
distance:74.7413787842 cm
distance:74.7413787842 cm
```

图4–16　程序执行结果4

靠近墙壁时，数值发生变化，如图 4-17 所示。

```
pi@raspberrypi: ~/maker_source_codes/PythonForMegaPi/examples
distance:58.189655304 cm
distance:58.1034469604 cm
distance:58.189655304 cm
distance:58.189655304 cm
distance:58.0862083435 cm
distance:58.189655304 cm
distance:58.2068977356 cm
distance:58.2068977356 cm
distance:58.2068977356 cm
distance:58.2068977356 cm
distance:58.2068977356 cm
distance:58.0862083435 cm
distance:58.189655304 cm
distance:58.189655304 cm
distance:58.2068977356 cm
distance:58.0862083435 cm
distance:58.189655304 cm
distance:58.189655304 cm
distance:58.0862083435 cm
distance:58.189655304 cm
distance:58.0862083435 cm
distance:58.0862083435 cm
distance:58.189655304 cm
```

图4–17　程序执行结果5

结果很明显，距离缩小。

程序说明如下。

```
def ultrasonicSensorRead(self, port, callback)
```

其中，port 表示连接到 MegaPi 上的端口号；callback 为回调函数，当采集到信号时调用的函数。

4.2.5　Me PIR Motion 传感器

Me PIR Motion 传感器是一个可以检测人体或者动物身上发出红外线的模块，它的最大检测距离为 6m。

连接方式如图 4-18 所示。

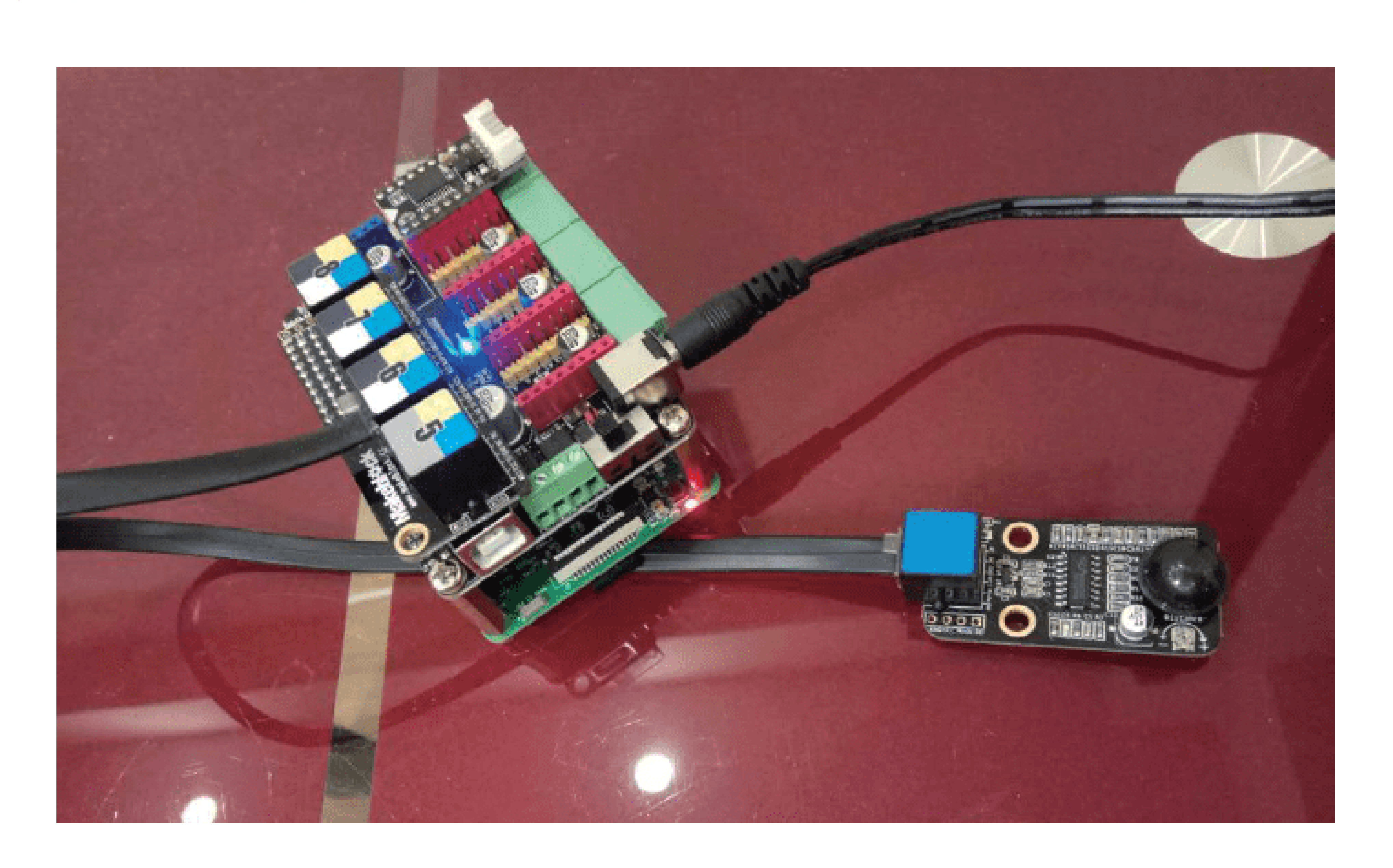


图4–18　Me PIR Motion与MegaPi的电路连接

该模块上有蓝色的标记，因此需要连接 MegaPi 上具有蓝色标记的端口。

人体红外线检测的 Python 代码如下。

```
from megapi import *
import signal
def onRead(v):
    print("find:", str(v))
def sigint_handler(signum, frame):
    global is_sigint_up
    is_sigint_up = True
    print "Exit!"
signal.signal(signal.SIGTERM, sigint_handler)
```




```
signal.signal(signal.SIGINT, sigint_handler)
signal.signal(signal.SIGHUP, sigint_handler)
is_sigint_up = False
if __name__ =='__main__':
    bot = MegaPi()
    bot.start('/dev/ttyS0')
    while 1:
        bot.pirMotionSensorRead(6,onRead)
        if is_sigint_up:
            break
```

运行结果如图 4-19 所示。

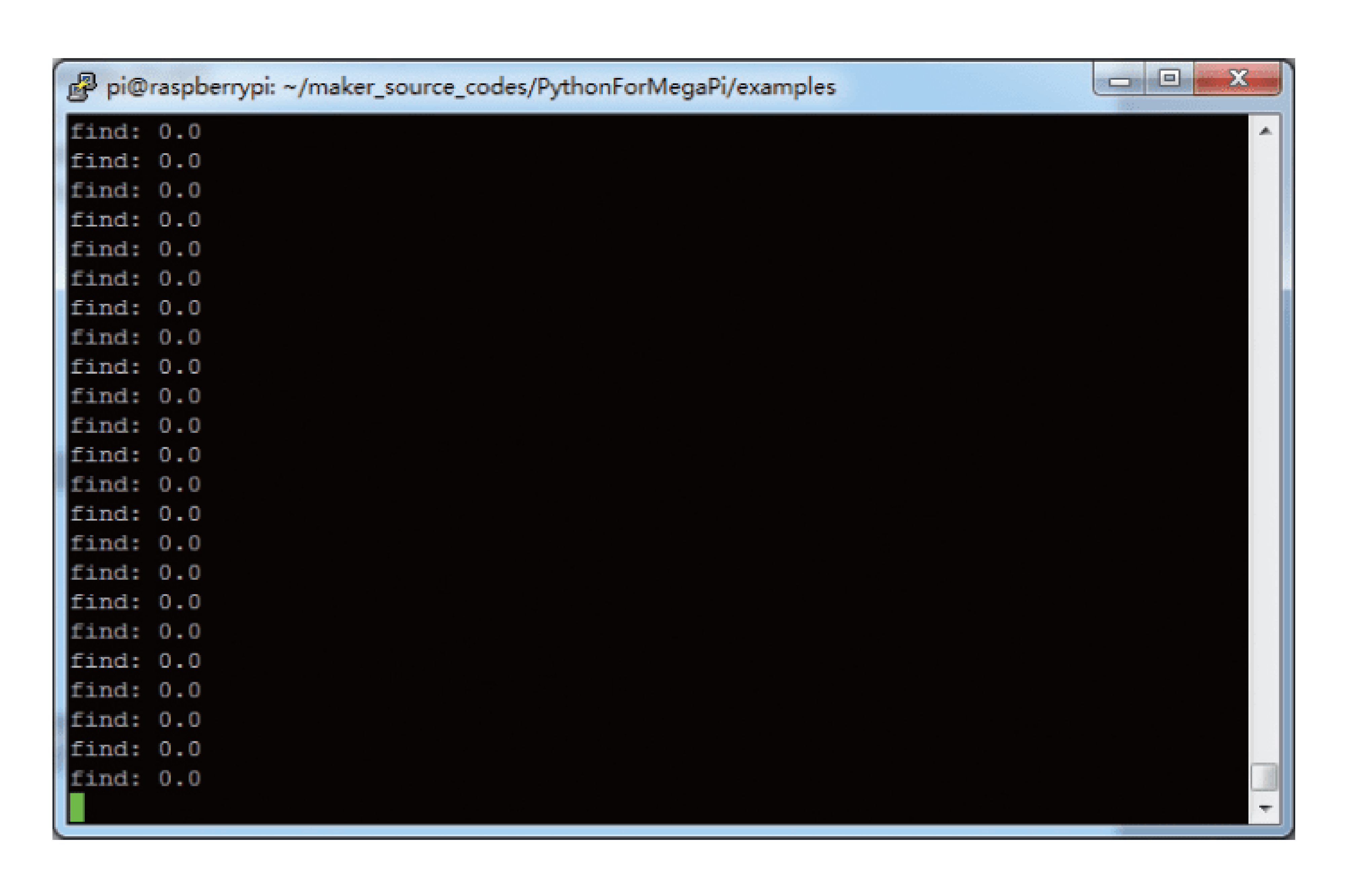


图4-19 程序执行结果6

手臂接近该传感器，运行结果如图 4-20 所示。



图4-20 程序执行结果7

可以看到数值变为 1，表示检测到人体的红外线信号。

程序解释如下。

```
def pirMotionSensorRead(self, port, callback)
```

其中，port 表示连接到 MegaPi 的端口；callback 为回调函数，输出为 0 时，表示没有检测到红外线信号，输出为 1 时，表示检测到人体红外线信号。

4.2.6　电位器

电位器是具有三个引出端、阻值可按某种变化规律调节的电阻元件。电位器通常由电阻体和可移动的电刷组成。当电刷沿电阻体移动时，在输出端即可获得与位移量成一定关系的电阻值或电压。图 4-21 和图 4-22 是两种常见的电位器模块。



图4-21　旋钮式电位器

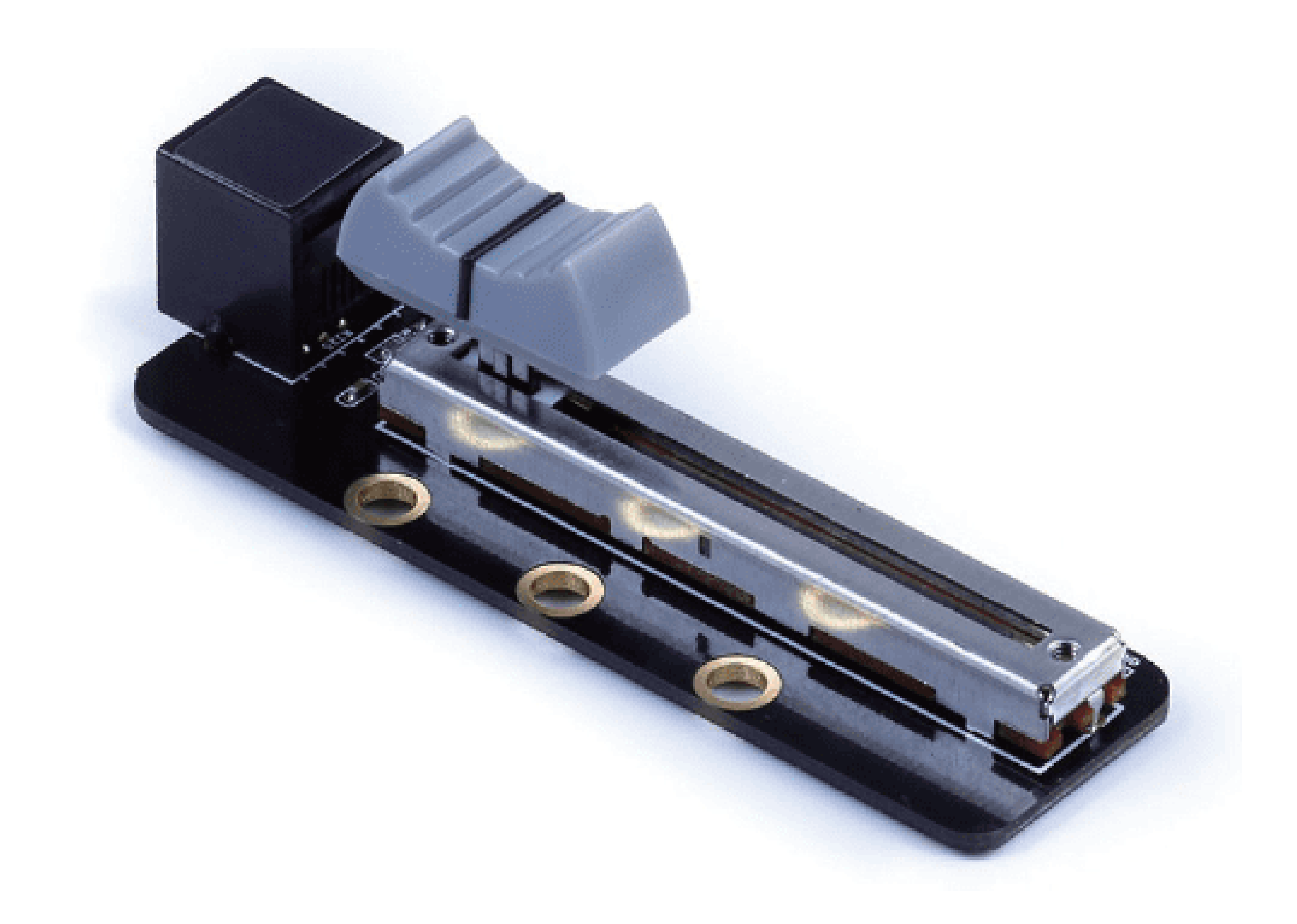


图4-22　直滑式电位器

电位器读值监测的 Python 代码如下。

```
from megapi import *
def onRead(v):
    print ("value:",str(v))
bot = MegaPi()
bot.start('/dev/ttyS0')
port = 6
while True:
    sleep(0.1)
    bot.potentiometerRead(port,onRead)
```

电位器的返回值范围为 [0,1023]。

4.2.7 角度传感器

角度传感器是用来检测角度的。传感器上有一个孔，可以插入轴，检测轴的旋转角度。Makeblock 角度传感器如图 4-23 所示。

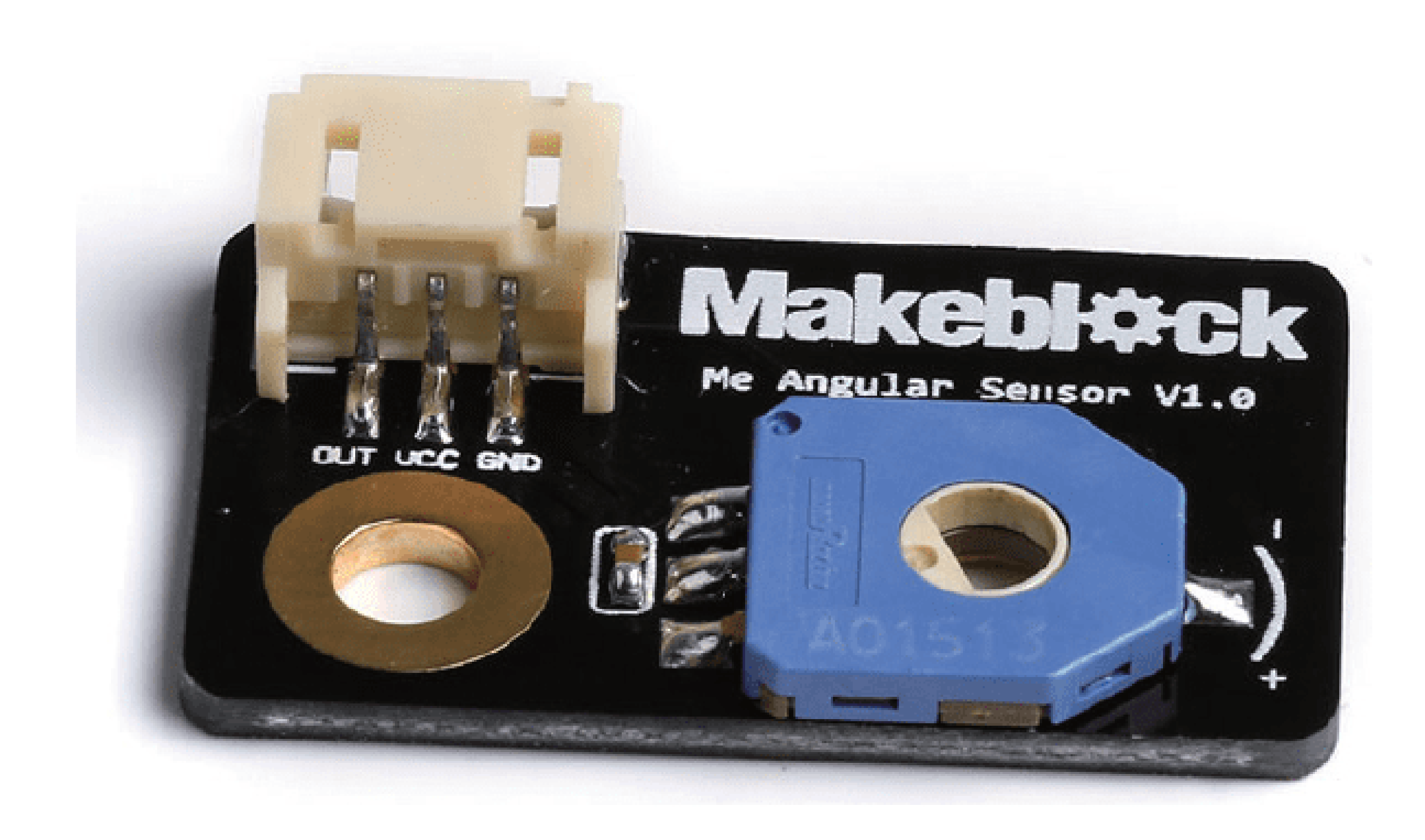


图4–23 Makeblock角度传感器

角度传感器读值监测的 Python 代码如下。

```
from megapi import *
def onRead(v):
    print ("angular:",str(v))
bot = MegaPi()
bot.start('/dev/ttyS0')
port = 6
slot = 1
while True:
    sleep(0.1)
    bot.angularSensorRead(port,slot,onRead)
```

角度传感器的返回值范围为 [0,1023]。

4.2.8 摇杆传感器

摇杆传感器模块用来控制小车的行走方向，也可以用来作为游戏的操纵杆。操纵杆可以被视为一个按钮（*Z* 轴）和电位计（*X*、*Y* 轴）的组合。连接方式如图 4-24 所示。



图4-24　摇杆传感器

模块上面的黑色标记表明需要连接到 MegaPi 上带有黑色标记的端口。

控制小车行走方向的 Python 代码如下。

```
from megapi import *
import signal
def sigint_handler(signum, frame):
    global is_sigint_up
    is_sigint_up = True
    print "Exit!"
signal.signal(signal.SIGTERM, sigint_handler)
signal.signal(signal.SIGINT, sigint_handler)
signal.signal(signal.SIGHUP, sigint_handler)
is_sigint_up = False
axisX = 0
axisY = 0
def onRead(v):
    global axisX;
    axisX = v;
    bot.joystickRead(port,2,onReadY)
def onReadY(v):
    global axisY
    axisY = v
    print (axisX, axisY)
if __name__ =='__main__':
    bot = MegaPi()
    bot.start('/dev/ttyS0')
```

```
port = 6
while True:
        bot.joystickRead(port,1,onReadX)
        if is_sigint_up:
                break
        sleep(0.1)
```

运行结果如图 4-25 所示。

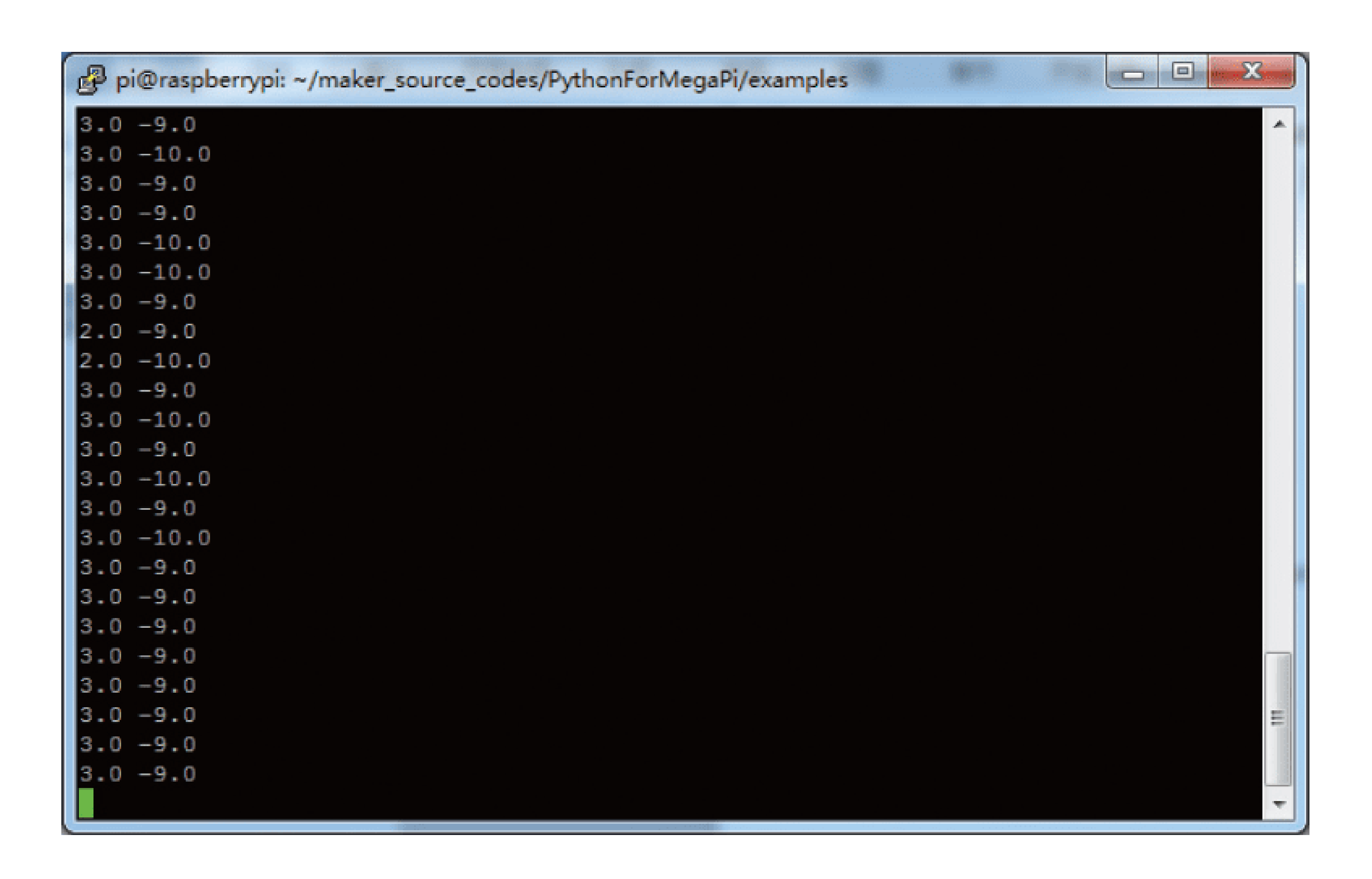


图4-25　程序执行结果8

转动一下操纵杆，可以看到 *X* 和 *Y* 的数值会随着操纵杆的转动而变动，如图 4-26 所示。

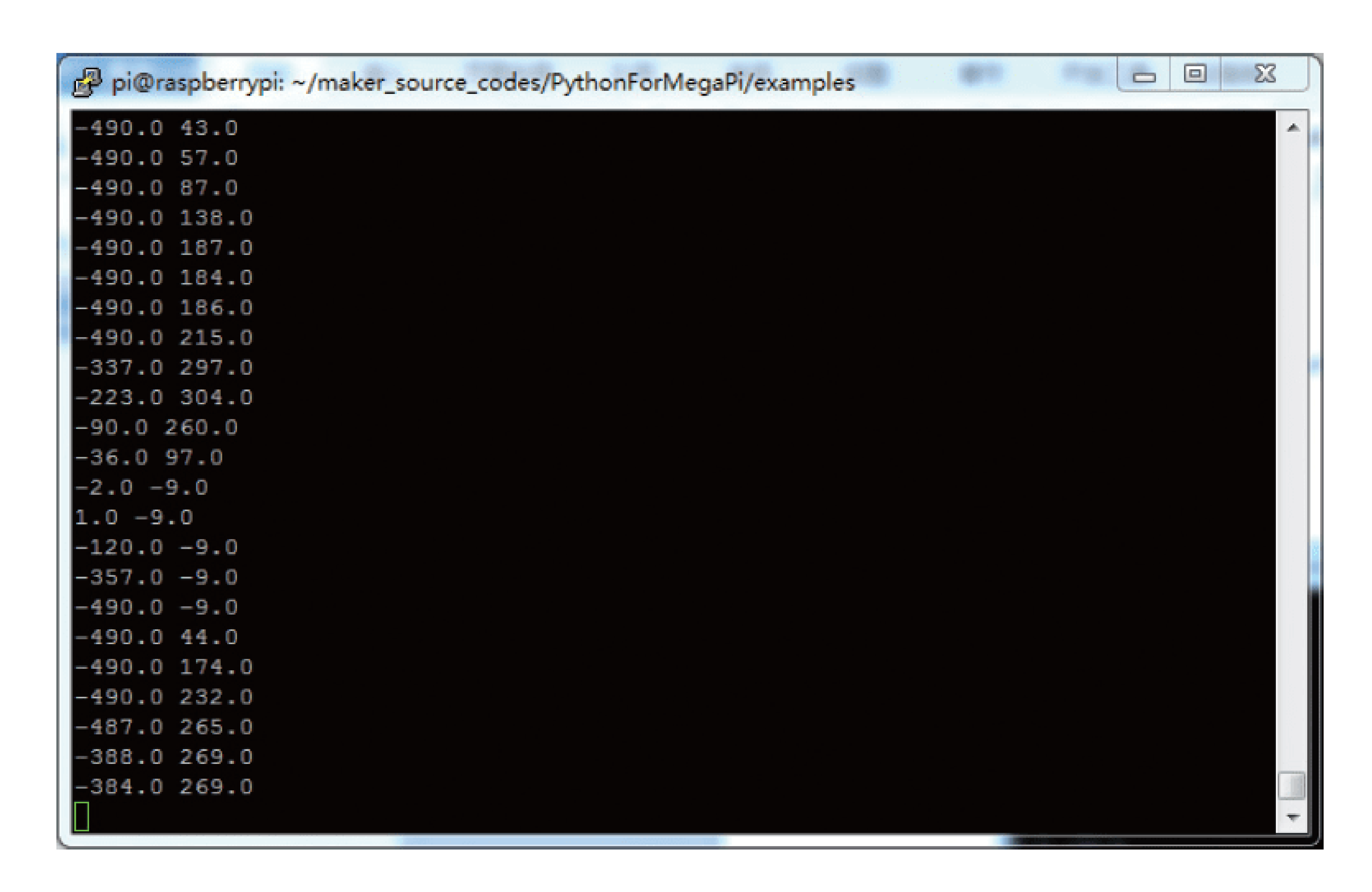


图4-26　程序执行结果9

4.2.9　巡线传感器

彩色传感器的工作原理是彩色 LED 发射一种单一色彩的光，遇到前方物体发生反射，当接触到不同颜色物体表面时，其反射回来的光的强度是不一样的，传感器会自行判断是否是所要探测的色彩信号，从而给单片机发送相应的信号，以达到控制要求。巡线传感器只需要判断黑白两色，可以只发射红外光，黑色的反射率低于白色的反射率，从而区分黑色和白色。Makeblock 巡线传感器如图 4-27 所示。



图4–27　Makeblock巡线传感器

巡线传感器循迹示意 Python 代码如下。

```
from megapi import *
def onRead(v):
    if v == 0:
        print ("both are white")                  # 两个传感单元检测到白色
    if v == 1:
        print ("left is white, right is black")   # 左传感单元检测到白色，右传感单元检测到黑色
    if v == 2:
        print("left is black, right is white")    # 右传感单元检测到白色，左传感单元检测到黑色
    if v == 3:
        print("both are black")                   # 两个传感单元均检测到黑色
bot = MegaPi()
bot.start('/dev/ttyS0')
port = 6
while True:
    sleep(0.1)
    bot.lineFollowerRead(port,onRead)
```

4.2.10　检测光线强度

光敏传感器是对外界光信号或光辐射有响应或转换功能的敏感装置，电子器件主要是光敏电阻，它能感应光线的明暗变化，并输出微弱的电信号，通过简单电子线路进行放大

处理，从而实现信号输出。Makeblock 光敏传感器如图 4-28 所示。

光敏传感器读值监测的 Python 代码如下。

```
from megapi import *
def onRead(v):
     print ("level:",str(v))
bot = MegaPi()
bot.start('/dev/ttyS0')
port = 6
while True:
     sleep(0.1)
     bot.lightSensorRead(port,onRead)
```



图4-28　Makeblock光敏传感器

光敏传感器的返回值范围为 [0,1023]。

4.2.11　检测声音强度

声强传感器的作用相当于一个话筒（麦克风）。它用来接收声波，并将其转换成电压的波动信号，但不能对噪声的强度进行测量。Makeblock 声强传感器如图 4-29 所示。

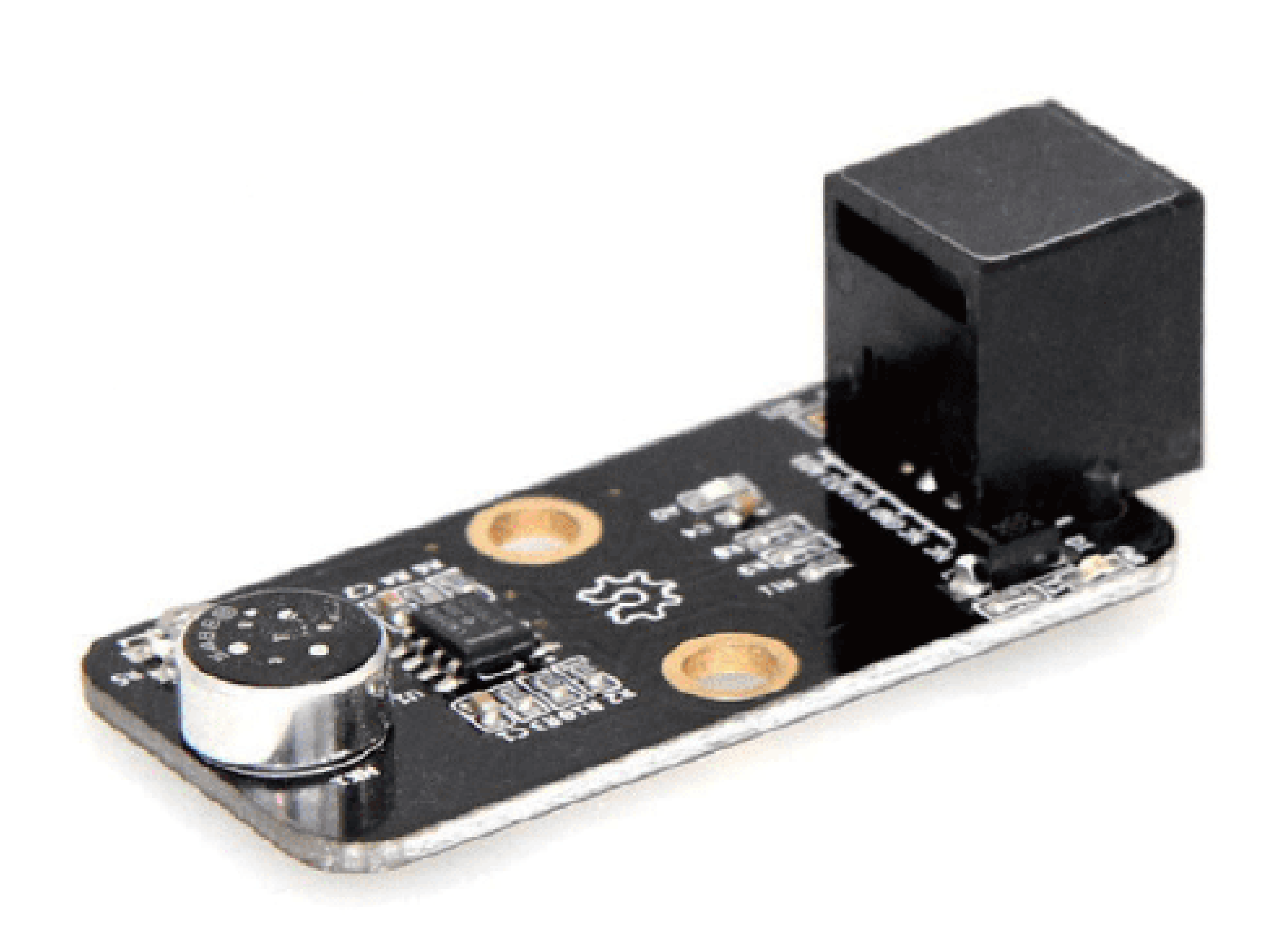


图4-29　Makeblock声强传感器

声强监测的 Python 代码如下。

```
from megapi import *
def onRead(v):
  print ("sound level:",str(v);))
bot = MegaPi()
bot.start('/dev/ttyS0')
port = 6
while 1:
  sleep(0.1)
  bot.soundSensorRead(port,onRead)
```

声强传感器的返回值范围为 [0,1023]。

4.2.12　检测温度和湿度

金属随着温度变化，其电阻值也发生变化。对于不同的金属来说，温度每变化一摄氏度，电阻值的变化是不同的，而电阻值又可以直接作为输出信号。

湿敏电阻的特点是在基片上覆盖一层用感湿材料制成的膜，当空气中的水蒸气吸附在感湿膜上时，元件的电阻率和电阻值都发生变化，利用这一特性即可测量湿度。

DHT11 温湿度传感器如图 4-30 所示。

温湿度监测的 Python 代码如下。

```
from megapi import *
def onRead(v):
     print("Temperature:"+str(v),"C")
     print "Humiture:"+str(v)+"%"
bot = MegaPi()
bot.start('/dev/ttyS0')
port = 6
while 1:
     sleep(1)
     bot.temperatureSensorRead(port,onRead)
     bot.humitureSensorRead(port,0,onRead)
```

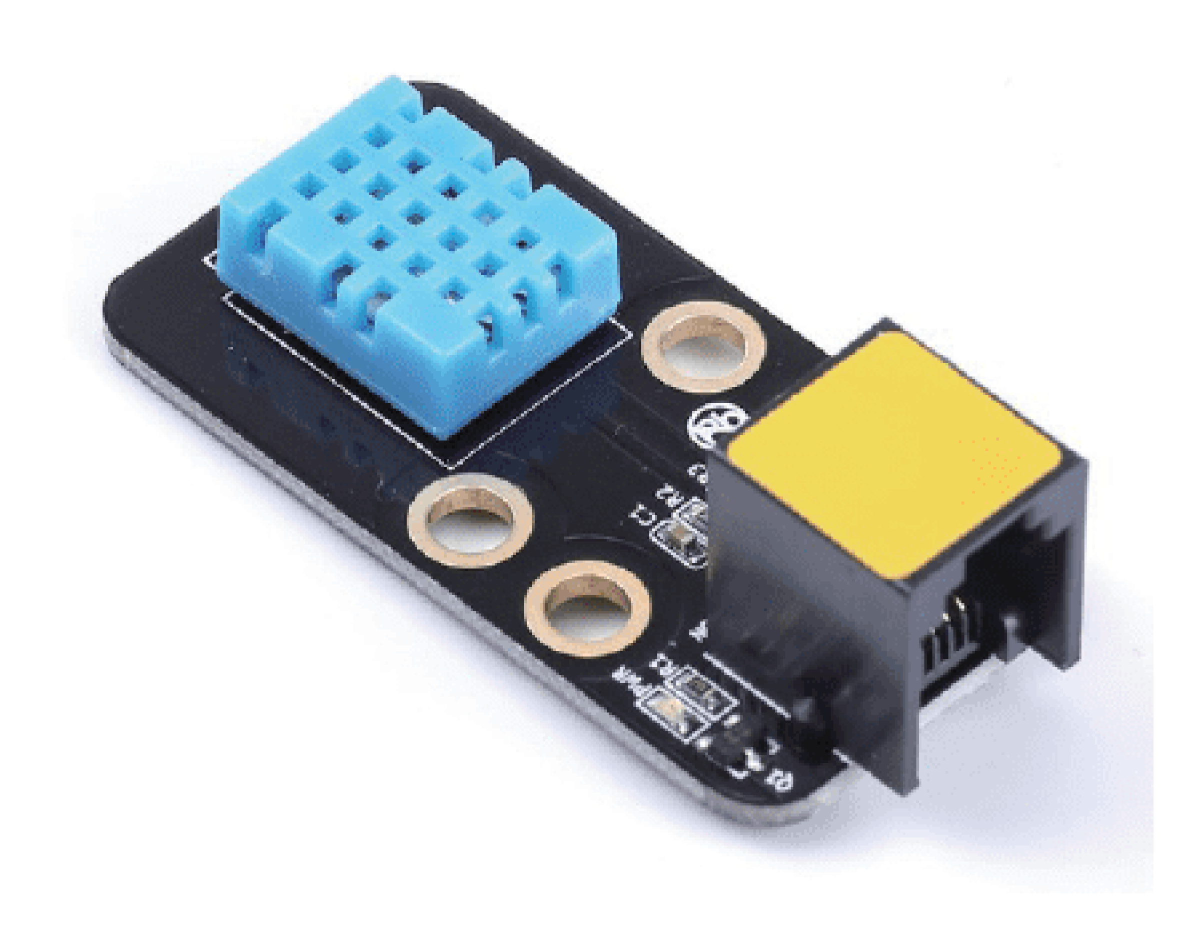


图4-30　DHT11温湿度传感器

4.2.13　检测手指触摸

所有电容式触摸传感系统的核心部分都是一组与电场相互作用的导体。在皮肤下面，人体组织中充满了传导电解质（一种有损电介质），正是这种导电特性，使电容式触摸传感成为可能。

触摸传感器如图 4-31 所示。

触摸传感器监测的 Python 代码如下。

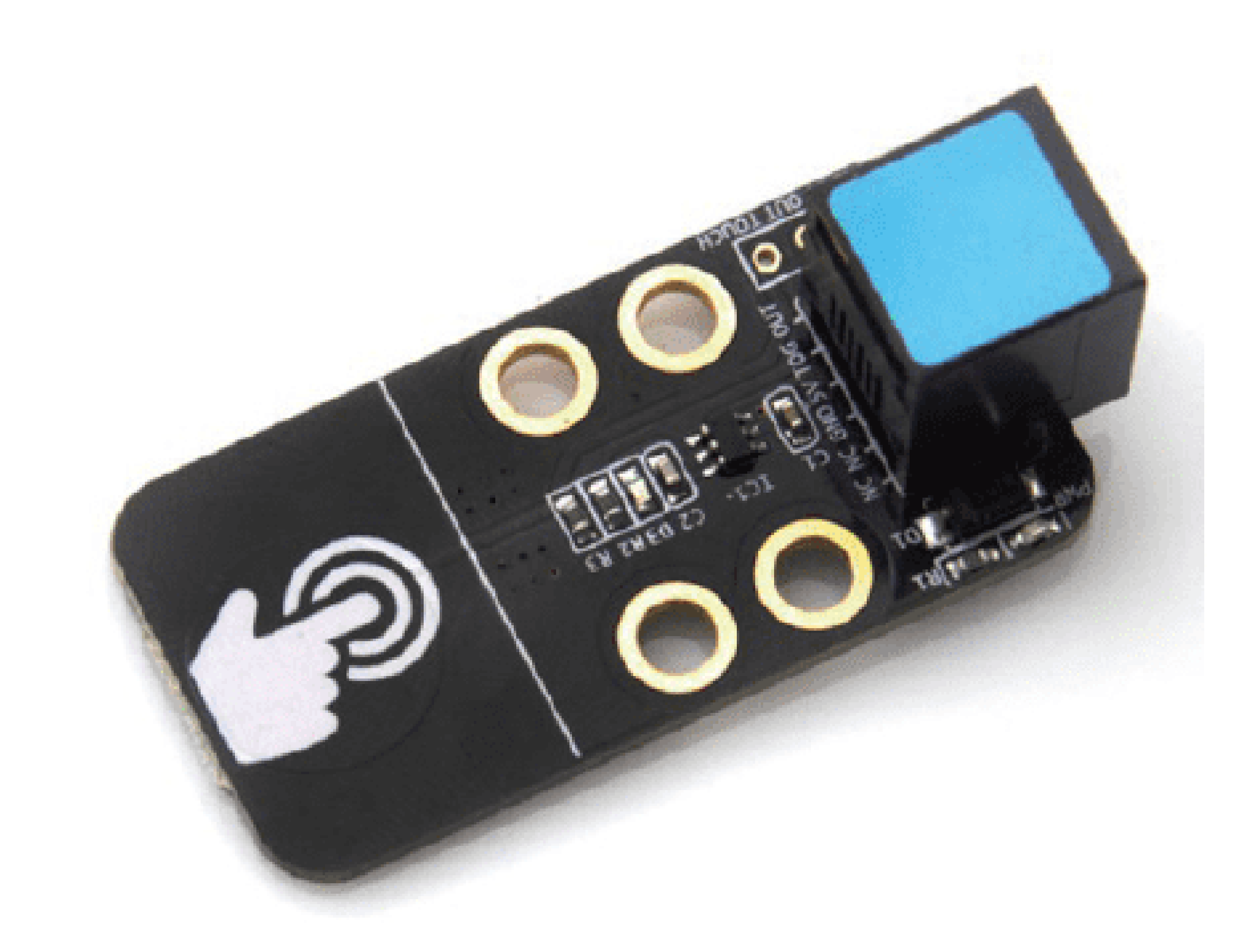


图4-31　触摸传感器

```
from megapi import *
def onRead(v):
  print ("touched:",str(v);))
bot = MegaPi()
bot.start('/dev/ttyS0')
port = 6
while 1:
  sleep(0.1)
  bot.touchSensorRead(port,onRead)
```

有手指触摸时，触摸传感器触点返回值为 1；否则，返回值为 0。

4.2.14　步进电动机控制

步进电动机是将电脉冲信号转变为角位移或线位移的开环控制元步进电动机件。在非超载的情况下，电动机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负




载变化的影响。当步进驱动器接收到一个脉冲信号，它就驱动步进电动机按设定的方向转动一个固定的角度，称为步距角，它的旋转是以固定的角度一步一步运行的。可以通过控制脉冲个数控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率控制电动机转动的速度和加速度，从而达到调速的目的。

步进电动机如图 4-32 所示。

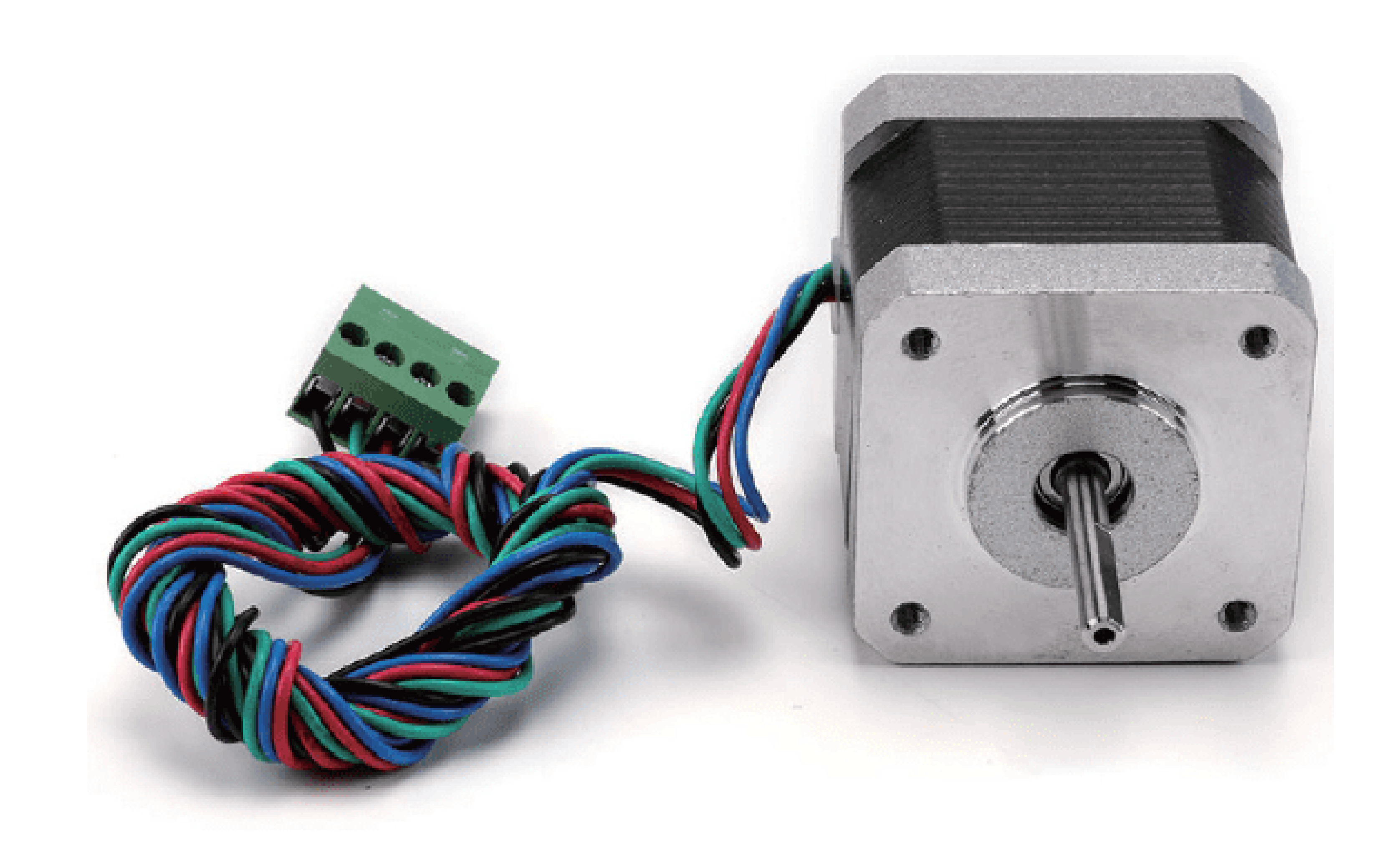
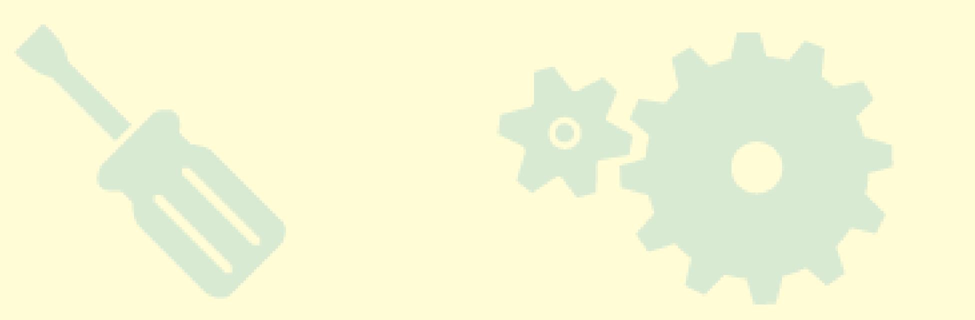


图4-32　步进电动机

步进电动机正 / 反转测试的 Python 代码如下。

```
from megapi import *
def onForwardFinish():
     sleep(0.3);
     bot.stepperMove(port,-2000,2000,onBackwardFinish);
def onBackwardFinish():
     sleep(0.3);
     bot.stepperMove(port,2000,2000,onForwardFinish);
if __name__ =='__main__':
     bot = MegaPi()
     bot.start()
     sleep(1)
port = 1
     bot.stepperSetting(port,4,5000);
     bot.stepperStop(port);
     onForwardFinish();
     while True:
            continue;
```



参考文献

[1] 董付国 .Python 可以这样学 [M]. 北京：清华大学出版社，2017.

[2] 张政桢 . 用树莓派去创造 [M]. 北京：清华大学出版社，2017.

[3] 深圳市创客工场科技有限公司 .https://www.makeblock.com[OL].[2019-1-25].